

A TRIDENT SCHOLAR PROJECT REPORT

NO. 465

Human Aided Reinforcement Learning in Complex Environments

by

Midshipman 1/C Carter B. Burn, USN



UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

This document has been approved for public
release and sale; its distribution is unlimited.

USNA-1531-2

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 05-21-18		2. REPORT TYPE		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Human Aided Reinforcement Learning in Complex Environments				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Burn, Carter B.				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Naval Academy Annapolis, MD 21402				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) Trident Scholar Report no. 465 (2018)	
12. DISTRIBUTION / AVAILABILITY STATEMENT This document has been approved for public release; its distribution is UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Reinforcement learning algorithms enable computer programs (agents) to learn to solve tasks through a trial-and-error process. As an agent takes actions in an environment, it receives positive and negative signals that shape its future behavior. To assist the process of learning, and to learn the task faster and more accurately, a human expert can be added to the system to guide an agent in solving the task. This project seeks to expand on current systems that combine a human expert with a reinforcement learning agent. Current systems use human input to modify the signal the agent receives from the environment, which works particularly well for reactive tasks. In more complex tasks, these systems do not work as intended. The manipulation of the environment's signal structure results in undesired and unexpected results for the agent's behavior following human training. Our systems attempt to incorporate humans in ways that do not modify the environment, but rather modify the decisions the agent makes at critical times in training. One of our solutions (Time Warp) allows the human expert to revert back several seconds in the training of the agent to provide an alternate sequence of actions for the agent to take. Another solution (Curriculum Development) allows the human expert to set up critical training points for the agent to learn. The agent then learns how to solve these necessary subskills prior to training in the entire world. Our systems seek to solve the planning requirement by employing a human expert during critical times of learning, as the expert sees fit. Our approaches to the planning requirement will allow the human expert-agent model to be expanded to more complex environments than the previous human systems developed. We hypothesize our project will increase the rate at which a reinforcement learning agent learns a solution to a specific task, and increase the quality of solutions to problems that require planning into the future, while successfully employing the use of a human teacher that guides the agents.					
15. SUBJECT TERMS reinforcement learning, agent teaching, advice taking, time warp, curriculum planning					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 81	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)

U.S.N.A. — Trident Scholar project report; no. 465 (2018)

**HUMAN AIDED REINFORCEMENT LEARNING IN COMPLEX
ENVIRONMENTS**

by

Midshipman 1/C Carter B. Burn
United States Naval Academy
Annapolis, Maryland

(signature)

Certification of Adviser Approval

Professor Frederick L. Crabbe
Computer Science Department

(signature)

(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

(signature)

(date)

Abstract

Reinforcement learning algorithms enable computer programs (agents) to learn to solve tasks through a trial-and-error process. As an agent takes actions in an environment, it receives positive and negative signals that shape its future behavior. To assist the process of learning, and to learn the task faster and more accurately, a human expert can be added to the system to guide an agent in solving the task. This project seeks to expand on current systems that combine a human expert with a reinforcement learning agent. Current systems use human input to modify the signal the agent receives from the environment, which works particularly well for reactive tasks. In more complex tasks, these systems do not work as intended. The manipulation of the environment's signal structure results in undesired and unexpected results for the agent's behavior following human training. Our systems attempt to incorporate humans in ways that do not modify the environment, but rather modify the decisions the agent makes at critical times in training. One of our solutions (Time Warp) allows the human expert to revert back several seconds in the training of the agent to provide an alternate sequence of actions for the agent to take. Another solution (Curriculum Development) allows the human expert to set up critical training points for the agent to learn. The agent then learns how to solve these necessary subskills prior to training in the entire world. Our systems seek to solve the planning requirement by employing a human expert during critical times of learning, as the expert sees fit. Our approaches to the planning requirement will allow the human expert-agent model to be expanded to more complex environments than the previous human systems developed. We hypothesize our project will increase the rate at which a reinforcement learning agent learns a solution to a specific task, and increase the quality of solutions to problems that require planning into the future, while successfully employing the use of a human teacher that guides the agents.

Keywords: reinforcement learning, agent teaching, advice taking, time warp, curriculum planning

Acknowledgements

I would like to thank all of those who have supported me throughout this journey. I would not be the student I am today without the amazing teachers I have had starting from my time in grade school. I'd like to thank my family for always believing in me and pushing me to excel and work harder. I'd like to thank my friends for putting up with my stress related to this project and the "many long nights and weekends" I had to give up to get this done. I am thankful for their support and understanding. Also, the Navy Powerlifting team has to get a shout out. Without the daily stress relief that the team brings me, I never would have finished this project. I'd also like to thank my advisors, Dr. Crabbe and Dr. Hwa. It was a pleasure working with both of you throughout this year and I am hoping our paper will get picked up by a conference. Thank you again for sharing with me your amazing knowledge and the dedication you both have to this field. Without that, I would not be submitting this paper. Lastly, and certainly the most important, I'd like to thank God for the opportunities He has blessed me with thus far in my short life. I would not be where I am today without Him.

Contents

1	Introduction	7
1.1	Motivation	8
1.2	Organization of this Report	9
2	Problem Statement	10
2.1	Problem Domain	11
3	Reinforcement Learning Background	14
3.1	Q-Learning	14
3.2	Exploration vs. Exploitation	18
3.3	Convergence and Optimal Policy	19
3.4	Representing the Q-Function	20
4	Improving Reinforcement Learning with Assistance	23
4.1	Assistance from a Teacher	23
4.2	Teaching Strategies	24
4.2.1	Human Expert Teachers	24
4.2.2	Computer Agent Teachers	26
4.3	Assistance through Scenario Presentation	29
5	Baselines and Experimental Setup	31
5.1	Frogger Q-Learning Baseline	31
5.1.1	Frogger Problem Set-up	32
5.1.2	Table-Based Q-Learning in Frogger	33
5.1.3	Q-Value Approximation	33
5.2	Computer Agent Baselines	35
5.2.1	Advise Important	36
5.2.2	Mistake Correcting	41
5.2.3	Ask Important-Correct Important	44
5.3	Comparison of Computer Systems	47
6	Human Teacher System - Time Warp	49
6.1	Time Warp Development	49
6.2	Experimental Setup	52

	4
6.3 Experimental Results and Discussion	53
7 Human Teacher System - Curriculum Development	60
7.1 Curriculum Development Design	60
7.2 Experimental Setup	63
7.3 Experimental Results and Discussion	64
8 Future Work	70
9 Conclusions	74
A Script Provided to Subjects	78
A.1 Time Warp Script	78
A.2 Curriculum Development Script	80

List of Tables

5.1	Experiments for the Advise Important System	37
5.2	Experiments for the Ask Important-Correct Important System	45
7.1	Curriculum Development Individual Trainer Results	67

List of Figures

2.1	Screenshot of Frogger	12
3.1	Q-Learning Example	18
3.2	Exploration vs. Exploitation	20
5.1	Feature Example	33
5.2	Q-Learning Baseline Results	35
5.3	Advise Important Results	38
5.4	Best Three Advise Important Experiments	38
5.5	Advise Important Best Experiments Individual Plots	39
5.6	Cumulative Attention for Advise Important	39
5.7	Comparison of Advise Important and Q-Learning	40
5.8	Mistake Correcting Results	42
5.9	Cumulative Attention for Mistake Correcting	42
5.10	Comparison of Mistake Correcting, Q-Learning, and Advise Important	43
5.11	Ask Important-Correct Important Results	45
5.12	Cumulative Attention for Ask Important-Correct Important	46
5.13	Comparison of Ask Important-Correct Important, Q-Learning, Advise Important, and Mistake Correcting	47
5.14	Best Computer Agent Teaching Systems	48
6.1	Time Warp Average Episode Reward	53
6.2	Time Warp Against Computer Systems	54
6.3	Time Warp Cumulative Attention Against Computer Systems	55
6.4	Early and Late Trainers Comparison	56
6.5	Good and Bad Trainers Comparison	57
6.6	All Human Trainer's Average Episode Reward	58
7.1	Curriculum Development 2000 Episode Continuation	64
7.2	Curriculum Development 5000 Episode Continuation	65

Chapter 1

Introduction

Writing software to solve complex problems is often difficult or impossible for people. For many problems, the potential number of situations to account for is too large for a programmer to anticipate. The field of machine learning (ML) has been developed to ameliorate these problems by enabling a computer program to learn directly from experience exploring the problem [8]. Within the ML paradigm, there are several techniques that may be applied, depending on the definition of the problem and desired outcome. Some problems require the classification of certain data, while other problems seek to teach a computer how to perform a specific task. One particular technique to teach a computer how to solve a given problem is known as reinforcement learning (RL), where the program learns how to map situations to actions to maximize a numerical reward signal [13]. During exploration of the problem, a machine's RL algorithm, or agent, will construct a function known as the policy. The policy function receives a state—a representation of the world—and outputs the action the machine should take in that state.

Eventually, a machine needs to explore the problem space—the domain in which the machine operates—enough to find an optimal policy, a policy function that maximizes the expected future reward signal with each action outputted [6]. One disadvantage of RL is that it can often take a long time for an agent to learn a reasonable policy as it explores large problem spaces, especially infinite problem spaces. Taking actions that produce zero or negative reward also increase the time it takes the agent to learn an optimal policy. RL

can be improved in a number of ways to speed up training and create better policies.

One improvement to RL systems is the addition of some form of assistance. The assistance can be introduced in a number of ways, but this report focuses on two main forms of assistance: advice from a teacher (either a computer agent or human expert) and creation of interesting and challenging scenarios to learn more about a given problem (either computer generated or human generated scenarios). These forms of assistance provide guidance to the agents to find the best policy within the problem space and possibly at a faster rate. This project investigates two state-of-the-art assistance systems by expanding on computer-based assistance methods and theories. One system, titled Time Warp, utilized assistance from a human expert by allowing the human to provide information to the agent at critical moments and proved to find a better optimal policy at a faster rate than a traditional RL algorithm. The second system, titled Curriculum Development, provided assistance by utilizing human generated scenarios to present challenging and unique training scenarios that proved to aid the agent in developing a better, generalized policy. This policy degraded over time, but proved at some points to perform better than any agent seen in the problem space.

1.1 Motivation

ML has helped make the world we live in a better place. ML algorithms assist cars in learning how to drive themselves, Amazon suggest products, and computers translate human languages more precisely. The field has developed sophisticated automation techniques without explicitly programming systems to account for every possible solution. The addition of assistance makes the learning more directed (and therefore, less random) which allows the machine to learn how to solve its task at a faster rate. The assistance method also enables the agent to use a human expert for important insights to the machine, such as safety and possible consequences of exploring the space; something that current models cannot grant. Traditional RL algorithms alone are only concerned with exploring a space more deeply in pursuit of solving the problem, not necessarily concerning itself with the consequences some actions may have.

However, the addition of a human expert to the RL process has been a difficult problem. It is desirable to have such a system to exist, but finding the proper interaction necessary for the complex RL algorithm to understand and acknowledge human input has been challenging. This project seeks to move forward from the current status of human-aided RL. With proper system design and human interaction that will not interfere with the problem space, this goal can be achieved.

1.2 Organization of this Report

This report will begin by defining the problem we seek to solve with our new systems. Next, we will introduce the necessary RL background and work that has inspired and formed a basis for this project. We will also explain how we implemented this same work in our specific problem space in order to compare the results to systems that utilize a human expert. Next, we introduce the two novel systems and the results of incorporating a human expert into these systems. Finally, we discuss the results and present expansions of these systems to make them perform better in the future.

Chapter 2

Problem Statement

Currently, the primary impediment to RL algorithms is the time it takes to converge on the optimal policy, or the optimal mapping of states to the correct actions in order to maximize future reward signals. This issue is apparent in all ML applications that have large or a possibly infinite problem space, and multiple adequate solutions. ML and RL algorithms alike have been refined to attempt to mitigate this problem, however the issue still exists in many applications. Due to the current demand for ML in everyday life, this problem stands at the forefront of current ML and RL research.

One possible solution to this problem is to use an entity that already has a solution to the particular problem. With many ML applications, humans typically know what the correct answer is, at least to an acceptable level. The introduction of ML to these problems simply allows the computer to find potentially better and more repeatable solutions than a human. If humans could assist an ML or RL agent in its quest for a solution, it is possible the overall training of the agent could take considerably less time overall. As enticing as this solution sounds, it is difficult to enable a human to assist in the training of an RL agent. Many computer scientists do not fully understand RL algorithms, much less people with no programming experience. For a human to properly assist an RL agent in training, the necessary mechanisms must exist that allow the human to fully interact with the system without specific knowledge of the workings of the RL algorithm, while also not interfering with the traditional training that the agent undergoes. This problem inherently defines the

goal of this project.

This project aims to create and evaluate novel methods of human interaction with a learning agent. These methods assist the training of the agent, while not destroying the internal mathematical representation of the agent’s problem space. With the proper utilization of human input, one of the dominant problems in RL can be mitigated: the time it takes to train an agent. Additionally, other forms of human assistance can be implemented to improve the policy of an agent.

To be able to determine the success of our methods, we must compare to a number of baselines. In particular, we must demonstrate that when using our methods, our human-assisted agent outperforms a standard RL algorithm in one or two ways: score (or amount of reward) and/or time. If the human-assisted agent can be more successful in less time or improve the policy of an agent, then we have shown our human interaction methods have solved the current problem.

2.1 Problem Domain

The domain in which an RL agent is attempting to find a solution to is the problem space. For example, a board or video game could be a problem space for an RL agent. The RL agent is attempting to “solve,” or win, the game. Many RL agents have found adequate solutions to famous games: like Pac-Man or Pong. In order to properly evaluate our human-based methods, we sought to find a domain that was difficult for humans to play, that has not been completely solved in academia, and that requires more planning, or foresight, than standard domains. For this reason, we chose the classic video game by Konami, Frogger.

In Frogger, the player is presented with a board that has a Frog (the player controlled character) at the bottom of the screen. The object of the game is to have the Frog make its way across the board to one of five homes at the top of the screen. Once the player places a Frog in a home, a new Frog spawns at the bottom of the screen to repeat the process. Once a Frog has been placed in all five homes at the top, the game is over. The challenging part is the objects along the Frog’s paths to its homes. First, the Frog must cross the road, which

has a number of cars crossing the screen in alternating directions and speeds, organized in five rows. Each row has either one, two, or three cars traveling in a line. Once the Frog has crossed the road, it must cross the river. The river consists of two objects also organized in rows: logs and turtles. Each row will either be a row of logs or a row of turtles and within each row the objects travel at the same speed. Each row also consists of varying numbers of logs or turtles, similar to the road. The Frog must hop between each row of logs or turtles without falling into the river. One important characteristic about the turtles is that some can dive and disappear from the screen. The Frog cannot jump onto diving turtles and must maneuver around them. However, for this project, the diving turtles were turned off. Finally, once the Frog reaches the last row of logs in the river, the Frog must hop into an open home. The Frog will die if it attempts to hop into the bushes near the homes or if the home is filled with a Frog. Homes also will randomly have flies or crocodiles spawn in them. Flies provide a bonus to the game score while crocodiles will eat and kill the Frog. Overall, the Frog can die in five ways: crashing into a car, falling into the river, jumping onto the bushes near the homes, or jumping into a home with a Frog or crocodile placed in it. Figure 2.1 provides a screenshot of the Frogger domain.

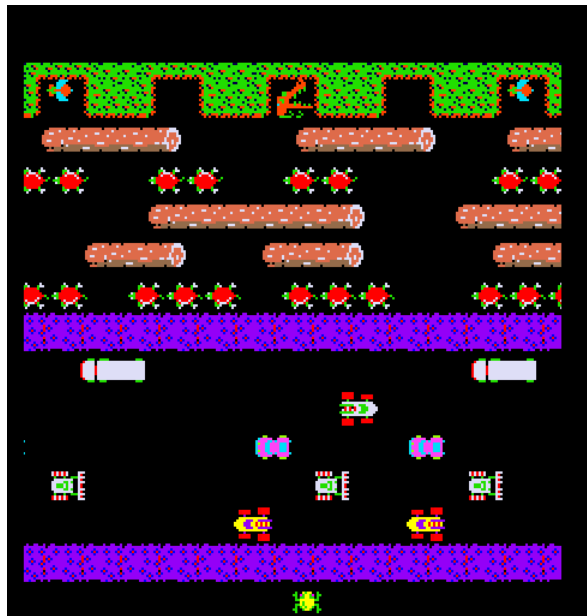


Figure 2.1: Screenshot of the domain used for this project, Frogger by Konami.

Frogger introduces some interesting aspects that are challenging to a standard RL agent. Particularly, the planning required to become successful at Frogger is enormous. For example, knowing which home to place the next Frog in is very important, as every subsequent move requires this knowledge to select the right direction to move. This necessary forward-thinking is difficult for RL algorithms and requires special planning to be able to encode such information. Additionally, this domain presents a large number of sub-problems that require very specific actions to properly solve. Subsequently, the agent must also learn that for half of the Frogger problem, the agent must avoid objects, while the other half of the problem the agent needs to jump onto objects only. Lastly, this domain also provides an environment where humans at least have some knowledge of success in the domain. Most humans have at least seen the concept of a Frogger-like game before, which allows for a larger number of human experts that can interact with the system. Frogger provides an excellent challenge to an RL agent, therefore if we can develop the proper human systems to provide assistance while training, we can prove that a more complex task can be solved. Solving a complex task in a fast and efficient manner would be phenomenal proof of our systems working as intended.

Chapter 3

Reinforcement Learning Background

There are many RL algorithms that each have specific properties in solving the task at hand. We will discuss the most common algorithm (and the one chosen for this project), Q-Learning. Additionally, we will present how recent work has used Q-Learning with the addition of a teacher to guide the learning agent to a solution faster than traditional algorithms alone.

3.1 Q-Learning

In the basic Q-Learning model, an agent selects actions based on the current world state and its *policy*, a mapping from states to the action to be taken in a given state. Q-Learning relies on a number of specific mathematical definitions and an algorithm to find the optimal policy, which is presented here.

A state, s , within the problem space is a representation of the current environment. A state could be a set consisting of such elements as a location, distances to a nearby objects, and other possible features that describe the world relevant to the agent. Every state in the problem space, $s \in S$, has a set of possible actions the agent may take in the state, $a \in A$. A state-action pair is a state paired with one of the possible actions. Each state-action pair will have a value assigned to it known as the Q-value, defined as a function $Q(s, a)$. The Q-value for each state-action pair represents the discounted expected reward, or the environment's

mathematical representation of success or failure, for taking action a in state s [6].

A Q-value is defined as

$$Q(s, a) = \sum_{s'} T(s, a, s') [R(s') + \gamma V(s')] \quad (3.1)$$

The transition function, $T(s, a, s')$, returns the probability that taking action a in state s will place the agent in state s' . $R(s')$ is the reward received for entering state s' . The discount factor, γ , is a human tunable parameter, or hyperparameter, that discounts the value of the next state. The Q-value is the sum over all possible next states from state s of the transition function, $T(s, a, s')$, multiplied by the reward of the next state, $R(s')$, plus the discounted value of the next state, $\gamma V(s')$. Discounting the value of the next state is done by selecting a discount factor such that: $0 \leq \gamma \leq 1$. A low γ means the agent does not take into account reward in the future, but only seeks to maximize immediate reward in the next state. A high γ means the agent cares more about future reward rather than immediate reward and will take actions that will get it closer to future reward. The value of the discount factor can be found experimentally and depends on the reward structure of the problem space to properly determine the discount factor. The value of a state, $V(s)$, is defined as

$$V(s) = \max_a Q(s, a) \quad (3.2)$$

The value function returns the maximum Q-value for a given state, or the value of the best action, which defines expected future reward of state s .

In small, discrete domains algorithms exist that can compute the exact Q-value for every state-action pair. However, expansion to larger domains restricts the computer's ability to compute every Q-value due to storage limitations. Additionally, some problems do not have the transition function ($T(s, a, s')$) explicitly defined. To solve this problem, the Q-Learning equations are still used, but the Q-value is approximated, rather than calculated it directly. This is done by having the agent explore the world and take samples of the reward received and the next state visited with a given action. When the agent takes an action in a state,

we define the sample as:

$$Q_{sample_t} = R(s') + \gamma V(s') \text{ where} \quad (3.3)$$

$$V(s') = \max_{a'} Q(s', a') \quad (3.4)$$

Once the agent transitions to the next state (s') the reward received and value of the next state can be calculated, thus providing the sample defined. By taking a running average of these samples, we can compute an estimate of the actual Q-value. Now, the algorithm can compute the Q-value without explicitly computing the transition function $T(s, a, s')$.

The Q-Learning algorithm evolves an optimal policy in four basic steps:

1. The Q-values for every state-action pair are initialized to either 0 or random values.
2. The agent takes an action based on its current policy (the function $\pi(s)$), taking the agent associated with the largest Q-value, defined as $\pi(s) = \arg \max_a Q(s, a)$. This results in the agent's receipt of a reward signal, plus a likely change in the state of the domain.
3. The Q-value for the previous state-action pair is updated by an amount proportional to the difference between the *previous* Q-value and the new information received from the Q_{sample_t} .
4. The policy is updated for the previous state based on the maximum Q-value for the state.

The algorithm computes the estimate by taking a running average with an update rule:

$$Q_{t+1} \leftarrow (1 - \alpha)Q_t(s, a) + \alpha Q_{sample_t} \quad (3.5)$$

This update rule is applied after every action the agent takes following the receipt of the reward signal and the agent's next state. By substituting equation (3.3) into the update

rule, we arrive at the equation:

$$Q_{t+1} \leftarrow (1 - \alpha)Q_t(s, a) + \alpha[R(s') + \gamma \max_{a'} Q(s', a')] \quad (3.6)$$

The running average taken to update the Q-values introduces another hyperparameter, α , the learning rate. This variable controls the computation of the running average and how much weight the algorithm gives to new samples. α must satisfy $0 \leq \alpha \leq 1$, and although the value is best derived experimentally, in order to guarantee convergence, it should decrease overtime.

By re-arranging Eq. (3.4) above, we can construct a more intuitive form of the update rule for Q-Learning:

$$Q_{t+1} \leftarrow Q(s, a) + \alpha[R(s') + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (3.7)$$

The construction of Eq. (3.5) provides a straightforward way to show how the Q-value is estimated throughout exploration of the space. The agent has a current estimation of the Q-value that it believes is wrong, $Q(s, a)$. The agent takes an action in the world and receives some reward, $R(s')$. The agent adds to the reward the discounted value of the next state it landed in, $\gamma V(s')$, or $\gamma \max_{a'} Q(s', a')$, to express the expected reward the agent will receive in the future from this point forward. The difference is taken between the current Q-value estimate and the sample. This difference expresses how wrong the current estimate of the Q-value is; the difference is then multiplied by the learning rate to move the estimate a small amount in the direction of the computed difference, as a single sample will not be the exact expression of the Q-value. The new Q-value estimate is thus a small change from the old estimate, slowly moving the Q-value towards convergence.

To provide a simple example of Q-Learning, consider the state space presented in Figure 3.1. There are four states and the agent can be in any of the four states. In each state, the agent can either take a left or right action. The agent will assign a Q-value to each state-action pair. Then, the agent will begin to explore the world and search for the state

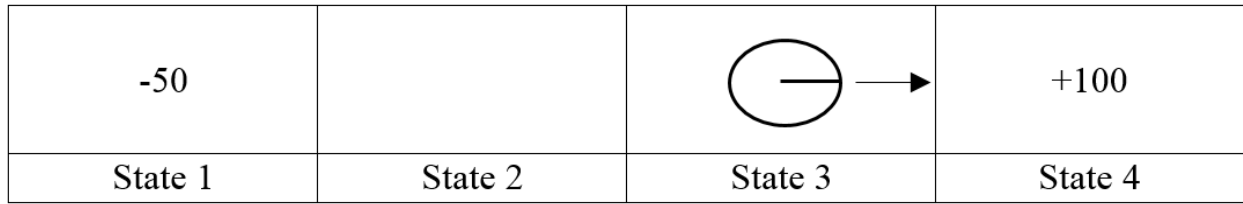


Figure 3.1: Q-Learning Example. This example shows an agent in state 3 about to take the rightward action to state 4, which is a state with maximum reward. This action will result in the Q-value for state 3, right action to increase due to the high reward in state 4. Over time, the value of $Q(3, a)$ will be the maximum Q-value for state 3, thus making the policy in state 3 map to the rightward action.

that provides the maximum reward (+100), while avoiding the state with negative reward (-50). As in Figure 3.1, the agent receives the maximum reward if it moves to the right in state 3. If the agent moved right, the agent would increase its Q-value for the rightward action in state 3, due to the positive reward received in state 4. After sufficient exploration, the rightward action in state 3 would have a larger Q-value than the leftward action, which means the optimal policy would direct the agent to go right in state 3, $\pi(3) = \text{right}$. If the agent took the rightward action in state 2 after state 3's Q-value was updated, the Q-value of the rightward action in state 2 would also increase. Although no reward was received, the discounted value of state 3, $\gamma V(3) = \gamma \max_a Q(3, a)$ is positive, thus causing an increase in the rightward Q-value in state 2.

3.2 Exploration vs. Exploitation

As mentioned, the algorithm takes samples throughout the world to estimate the Q-values throughout the space. The agent decides to take these actions based on the current Q-values, according to the policy $\pi(s)$ which defines the action associated with the highest Q-value of a given state. This could introduce a problem once some of the Q-values are partially developed, as the agent will only take the learned actions and never investigate other actions that could potentially lead to higher reward. In order to mitigate this problem, the agent can take random actions in the space, ignoring the developed policy. This is known as the

Exploration-Exploitation Tradeoff. While learning, the agent can either take exploration or exploitation steps. An exploration step is taken by selecting a random action to derive more information from the world, while an exploitation step is taken to exploit the current policy, or take an action that maximizes the expected future reward. The algorithm (when selecting the next action to take) must properly balance which types of steps to take in order to explore the entire domain, but still find the maximum reward and reinforce the maximum reward in the problem space. This choice is controlled by another hyperparameter, ϵ . This parameter must satisfy: $0 \leq \epsilon \leq 1$ and represents the probability that the next action the agent selects is a random one. ϵ should fall throughout training, so that the developed policy can be properly reinforced and convergence can be reached.

Figure 3.2 provides a simple example of the Exploration vs. Exploitation Tradeoff. If the agent has developed a policy that provides a reward in one part of the domain, directing the agent toward that reward is exploiting the current policy. However, it is possible that an even larger reward exists within the problem space and if the agent continued to exploit the policy, it would never find this larger reward. With the introduction of exploration (or random) steps, the agent increases the chance that it will find this larger reward.

It is important to note the impact exploration has on an agent’s training. The correct exploration will lead the agent toward the highest reward, but this does not necessarily mean a higher ϵ will create “correct” exploration. The balance of exploiting the current policy and exploring unknown spaces in the domain leads to the uncovering of higher reward. Exploration can be the difference between a successful, average, and terrible learning agent. Additionally, exploration plays a key role in both systems developed in this project.

3.3 Convergence and Optimal Policy

The Q-values are said to have converged when there is little to no change in the update of the Q-values during repeated exploitation of the current policy. Another sign of Q-value convergence is when the exploitation of the policy consistently returns similar total reward for each test run. Once the Q-values for every state-action pair converge, the agent has

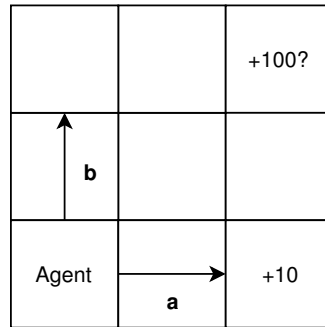


Figure 3.2: Exploration vs. Exploitation. Arrow a represents the agent taking an exploitation step toward the known reward of +10. Arrow b represents the agent taking an exploration step away from the known reward toward an unknown area of the problem space. The exploration step may or may not find a more generous reward, but would never know if it didn't take the step.

effectively finished learning. At this point, the agent has hopefully developed a near-optimal policy (the algorithm is guaranteed to eventually reach the optimal policy in infinite time, but usually converges before that). The policy will take the given state and map it to the action that has the highest Q-value among the state-action pairs for a given state. At each state, the optimal policy will instruct the agent to take the action that leads the agent toward maximizing future reward. In a given run of solely exploiting the policy, the optimal policy will guarantee maximum future reward with each outputted action from the function. The optimal policy is given by $\pi^*(s) = \arg \max_a Q^*(s, a)$.

3.4 Representing the Q-Function

There are two primary approaches to represent the function that maps states and actions to Q-values. The first is a table. A table-based Q-Learning approach is a “brute force” solution to Q-Learning, where the program stores the Q-value of every state-action pair in a large table in computer memory. Clearly, this method has its limitations, especially when the problem space becomes large due to a large number of states and possible actions. The number of Q-values needed for a given problem space is $n * m$ where n is the average number of actions for each state and m is the total number of states. This means the space

complexity for the table-based algorithm increases rapidly as n and m increase. Table-based Q-Learning is still possible, however, only in certain domains where the problem space is finite. Table-based solutions also typically take a much longer time to train and achieve convergence of the Q-values than other methods.

The other technique used with Q-Learning is to employ a Q-Value Function approximation. Effectively, the agent must first find an alternate representation of a state-action pair, represented with a set of features. Features extract meaningful and concise information from the current state in the world. Features assist in reducing the dimensionality of the problem space and allow for the use of an alternate approach to storing the Q-value. Features can be rudimentary, like the position of the agent in the world, or highly engineered, like calculating the number of enemies near the agent after taking an action. Every domain will have specific characteristics that can be represented with features and the feature set will likely change with every new domain.

With features, the Q-value for a given state-action pair can be a function of the features (that take the state and the action as input) and a set of weights that are assigned to each feature. Thus, each feature has a weight associated with it. The function chosen to represent the Q-value must correctly represent the mathematical nature of the problem, but the most common starting point is simply a linear combination of the weights and features:

$$Q(s, a) = \sum_i w_i f_i(s, a) \quad (3.8)$$

$f_i(s, a)$ represents the feature function for a given state-action pair. The function will return a numerical value that is multiplied by the weight associated with that feature and ultimately added to the sum of other weight-feature combinations for the function approximation of the Q-value. Since the Q-value definition has changed, the update rule for each action taken is slightly modified. With each step in training a difference is calculated:

$$\text{difference} = [R(s') + \gamma \max_{a'} Q(s', a')] - Q(s, a) \quad (3.9)$$

and each weight is updated according to the new update rule:

$$w_i \leftarrow w_i + \alpha[\text{difference}]f_i(s, a) \quad (3.10)$$

If the problem space is not strictly linear, then other functions of the features may be used to properly represent the Q-value.

Chapter 4

Improving Reinforcement Learning with Assistance

While Q-Learning can be used to develop policies for given tasks, the development of such policies may take a long time. One technique to increase the speed of policy development is to add assistance to the training. Assistance can be added in a number of ways, but this report will focus on two types of assistance: advice from a teacher and the creation of scenarios to increase exploration of the problem space.

4.1 Assistance from a Teacher

In the context of RL, a teacher is an entity with prior knowledge of the environment and task. By sharing this prior knowledge with the learning agent, the teacher helps to enhance the policy development of the learner, so that the time it takes to develop the policy can be reduced while also finding a better policy.

There are two types of RL teachers, computer agents and human experts. Both teacher types present unique challenges. Using a computer agent requires the teaching agent to already have developed an effective policy for the environment. The development of the computer agent's policy could be done by training a separate RL agent or by having a human expert hard-code the policy (a straightforward approach for easy tasks). It is important to

note that the use of a computer agent in this way inherently undermines the purpose of using a teacher. There is no reason to train another learning agent if an agent already exists that developed an effective policy. On the other hand, using a human expert presents its own significant challenges. A human expert needs to have an interface with the agent, a knowledge of the environment, and have enough skill to provide useful advice to the learning agent. An effective human teacher system must confront all of these issues in order to be an effective teacher to a learning agent.

4.2 Teaching Strategies

We divide the work using a teacher with Q-Learning into categories based on the type of teacher, either a human expert or computer agent.

4.2.1 Human Expert Teachers

Richard Maclin and Jude Shavlik formed a stable basis for human experts providing advice to learning agents. Their system, developed in 1996, first demonstrated the human addition to training and showed promising results. One primary difference was that they used a scripting language that allowed the human to programatically provide advice based on situations that the human programmed. This is similar to many systems, but the need to program advice is undesirable. Even simple programming languages can limit the amount of human experts available for a given problem. Ideally, any human with skills within the domain can provide advice, without knowledge of programming or reinforcement learning. Regardless, Maclin and Shavlik’s systems provide an excellent base to further the use of human experts in reinforcement learning [7].

Thomaz and Breazeal created an algorithm and an interface that incorporated a human teacher [15]. The interface, known as the *guidance framework*, allowed the human to provide feedback after each action, thereby combining the environmental and a human reward structure. Their system created a policy quicker than RL algorithms alone (in the same domain), but lost significant accuracy in the actual policy, never reaching an optimal policy. Instead,

it suffered from a credit assignment problem. The learning agent could not tell what action deserved the reward and thus did not learn the proper mapping of states to actions. Essentially, the human was providing guidance to the agent as opposed to feedback of the agent's actions. The human's intended results were not properly captured by the human-teacher interface and Thomaz and Breazeal's learning agent could not determine the optimal policy. In fact, the agent had to unlearn the policy the human expert taught to be able to learn the correct policy. Additionally, the learning system could not learn without the presence of the teacher, rendering the system useless without human guidance. Ideally, the learning system still could learn on its own and receive human assistance when the human expert desires. Thomaz and Breazeal's research did provide an important point concerning how a human typically provides assistance to an agent in training. Humans are more inclined to provide guidance to an agent rather than provide feedback. This is an important consideration we made while designing our systems.

Another system developed by Bradley Knox and Peter Stone, named the TAMER framework, also utilized a human expert assisting in the training of an agent [3, 4, 5]. To summarize the body of research Knox and Stone conducted, it is important to note their intentions. The TAMER framework is designed to allow an agent to learn a policy as defined by a human reinforcement (or reward) function, rather than an environmental reward function. The agent is learning how to solve a task that it cannot directly measure, but the human expert can. The human expert has their own internal definition of success in the given domain and utilizes its own sensory display to determine whether the agent is succeeding or not. Then, the human provides reinforcement (a positive or negative signal), which the agent uses to update its internal Q-value calculation. Additional work with the TAMER framework included adding a probabilistic error buffer for when the human provided its reward signal to determine which action to attribute the signal. This allowed the credit assignment problem seen in Thomaz and Breazeal's work to be substantially mitigated. The TAMER framework performed exceptionally well in its experiments. TAMER demonstrated that a human expert can be utilized to train an agent that is learning how to solve a task that does not have an explicitly defined environmental reward function, but rather a reward

structure only defined by a human’s sensory reactions to the agent’s actions. The agent, then learning according to the human’s reward model, builds its own learned model representing (and hopefully mimicking) the human teacher’s model. While Knox and Stone’s work is important to note for our systems, it is not directly related to our work. We are not designing a system that will learn a human reward function based on human feedback, but rather systems that attempt to learn an environmental reward with the assistance of a human. We are not designing agents to exploit human-defined reinforcement functions, but rather utilizing human expertise to help the agent learn the environment’s reward structure in order to maximize expected reward within the environment.

Moreno *et al.* introduced another system, titled Supervised Reinforcement Learning (SRL). SRL attempted to allow a prior knowledge source (PKS) to provide advice to a training agent. This PKS will provide an action in a number of “representative” positions throughout the domain. Essentially, prior to training, a specified number of states must be determined as important training moments and the exact action is provided to the agent in each of these pre-determined important states. The PKS used in their experiments was a human expert in the domain. SRL proved that the agent arrived at convergence quicker because of the ability of the PKS to direct the proper exploration of the agent throughout the space. Again, another important consideration here was taken for our systems. We want to incorporate this idea of directing exploration, however our systems differ from the SRL systems in various manners. We are utilizing the human expertise while training is taking place, not prior to training. Additionally, we are not only providing a single action in important times in training, but rather a number of actions as determined by the human [9].

4.2.2 Computer Agent Teachers

Another form of a RL teacher is a computer agent that has already developed a policy for the given task. There are a number of ways to employ a computer agent as a teacher, but we will focus on the teachers that provided advice to the new agent learning the task.

Torrey and Taylor developed a system utilizing a computer agent in the Pac-Man video game. Their system provided *action advice* to the learning agent from the already trained computer teacher [16]. Advice provided by the teacher is simply the action to take in a particular state. The advice is provided to the student agent in select states, as defined by the algorithm that determines when advice is given. In Torrey and Taylor’s model, the learner would only receive the advice in states that were defined as important when comparing the importance calculation of a state to a certain threshold. Torrey and Taylor defined the importance of a state as $I(s) = \max_a Q(s, a) - \min_a Q(s, a)$. If $I(s)$ was above a human-defined threshold value, the state was considered important and the computer agent would provide the action to take in the state. This model gave the teacher agent, with an already learned policy, the ability to advise the student in unexplored states and states that had certain positive or negative reward. Additionally, Torrey and Taylor experimented with another system that would only actually provide the advice when the student and teacher’s actions differed in the important states, thereby decreasing the amount of times the teacher is utilized. In both systems, the teacher was restricted by an advice budget. The budget is simply the number of times the teacher can provide advice. The budget was a hyperparameter to the system and once it ran out, the teacher would no longer provide advice. Torrey and Taylor’s system showed that *action advice* allowed the student learning agent to learn an effective policy at a much faster rate than a traditional RL algorithm and outperformed the majority of the human expert systems. The primary advantage of the computer agent is the ability to monitor the internal actions of the student. Also, the teacher provides instant advice, without the almost certain delay between a human observing an important state and deciding to provide advice. With these systems, advice provides a mechanism to encourage exploration of the space toward known reward, while the human systems provided a mechanism to provide reward or feedback to the agent. This difference between the two types of teachers is the reason that computer agents had more success than human experts. The utilization of proper exploration is an important design principle we used in our systems.

Amir *et al.* expanded on Torrey and Taylor’s work, creating a system called *apprenticeship*

learning in Pac-Man [1]. This system allows the student to ask for advice in important states, as opposed to the teacher providing the advice in important states. The student also uses its own Q-values to determine its own definition of important states, which means as the student learns, it will rely less on the teacher and not need advice as often. When the student believes it is in an important state it asks the teacher to check, if the teacher does not agree that the state is important, the student is given control to take its action based on its policy or an exploration action. If the teacher does believe it is an important state, the teacher will then check to see if the student action does not match the teacher's action in the current state. If the actions do not align the teacher provides advice in the state. The results of this system were similar to Torrey and Taylor in terms of how fast the policy was developed and quality of the policy developed. However, the number of times the teacher actually provided advice was significantly less than Torrey and Taylor's system. Amir *et al.*'s system showed that when the student asks for advice, the teacher is not needed as many times as Torrey and Taylor's system because the student improves as training continues and does not need to rely on the teacher's advice as often. This is an important idea to consider when designing a human teaching system, as human experts will not have the patience to train an agent for as long as a computer agent can.

With all of these teaching systems, there is still room for improvement. As mentioned earlier, the use of computer agents is redundant. If an agent with an effective policy already exists, there is no reason to train another agent. However, the clear success of computer agents means it is possible to improve human systems to work this way as well. The current human systems utilize differing techniques to incorporate the human into training. We plan to incorporate a human system similar to the methods the computer teachers utilized, while taking important considerations and lessons learned from previous human systems. Our system will focus on assisting the agent in exploration of the space toward areas of known reward and not utilize a feedback mechanism as seen in the other human teaching systems. We hypothesize that directing exploration of the space, as the computer agent teachers did, the agent will learn a more effective policy quicker than other human systems that controlled the reward the agent received.

4.3 Assistance through Scenario Presentation

Another form of assistance is scenario presentation. In this method, the agent is presented with scenarios of the task that can help the agent learn necessary sub-skills within the domain. This form of assistance can also be accomplished with a human expert or a computer that generates the scenarios.

Narvekar *et al.* developed a system called Curriculum Learning, where functions that can produce scenarios for an agent to learn and transfer their knowledge to the larger domain [10, 12]. Their work includes a number of methods to automatically develop scenarios while an agent is on a specific training trajectory. These methods include simplifying the overall task automatically, moving the agent near states with high reward, allowing an agent to reverse training after a death to correct mistakes, and even linking tasks together to create the overall task. Additionally, they developed a Markov Decision Process that can develop a sequence of these tasks that should be followed for a given agent. Some of these tasks require domain knowledge, while others are domain independent and they change how the agent trains. Overall, this work provides context for our system, Curriculum Development. We similarly are creating scenarios for the agent to learn, but these scenarios are actually created within the larger target task, as opposed to creating scenarios and tasks that are a subset of the target task. Additionally, these tasks are not created along a trajectory of learning, but rather created after an agent has developed an effective policy. Lastly, the most distinct difference is that we have a human expert create our scenarios, rather than another RL method.

While computer generated scenarios are the dominating research within this form of assistance, we do see the possibility to expand to a human expert providing scenarios. While a human may not be able to quantify a curriculum to develop like a computer can, humans have an intuitive idea of what tasks are required for an agent to learn an overall target task. Since the human expert will not be learning the curriculum to actually produce, but developing it for the agent, we titled our system Curriculum Development. The development of the scenarios that the agent is placed in is completely determined by the human’s own

idea of useful scenarios to place the agent in.

It's also important to note that the human's interaction will also effect other aspects of the training that Curriculum Learning did not. The human interaction will encourage exploration of the space, not necessarily creating tasks for the agent to learn during training. These scenarios will allow the agent to explore different areas of the problem space and eventually develop a more generalized solution to the task.

Chapter 5

Baselines and Experimental Setup

In this chapter, we replicate some related work and establish a baseline for comparison with our systems. First, we discuss the design of a basic Q-Learning agent that will run in the Frogger domain. Then, we discuss the replication of the most successful computer agent teacher systems in order to provide a comparison with our human systems.

5.1 Frogger Q-Learning Baseline

As discussed above, we chose Frogger as our environment because of the large number of sub-problems that are inherent within the domain. The planning element of Frogger also makes the environment inherently complex for standard RL algorithms. Also, since many humans have at least seen a version of Frogger before, it will provide a great environment to allow humans to assist in training an agent. In order to compare our system to previous ones, we needed to replicate their results in our domain.

Our first step in building the baseline for our systems was to build the Frogger domain. Although we found several modern implementations of Frogger, we found their design structure unamenable for the addition of learning systems. Instead, we reorganized one implementation of Frogger in *pygame* and rewrote the overall game code to better suit the integration of a learning agent. We used the graphics and object initialization written by the padorange user on SourceForge [11]. The game was written for a human player to use the

keyboard to control the Frog’s movements. Also, the logic for the scrolling cars, turtles, and logs was added, along with the creation of flies and crocodiles in the homes. The speeds and locations of all scrolling objects were derived from the source code of padorange’s Frogger implementation. Figure 2.1 provides a screenshot of the finished game.

In order to incorporate RL into our game, we used the PyGame-Learning-Environment (PLE). PLE provides an interface for RL agents and games written in *pygame* [14]. This additional Python module allows us to wrap our written Frogger game into PLE to allow an external agent to be written to control the movement of the Frog. PLE provided a seamless interface to the written Frogger game and the many RL agents we wrote for this project. PLE made the development of multiple experiments a much easier task.

5.1.1 Frogger Problem Set-up

Emigh *et al.* used Frogger in their work [2]. Their work was not directly related to ours, but did give a starting point for designing an agent for Frogger. We modified their local feature set for our state-space representation. The local feature set will only determine if a car, river object, or home is within a 3x3 grid around the Frog. Each square is a 32x32 pixel wide box. If an object lands within the box, the feature will “fire,” or be activated. Figure 5.1 provides an example of a car object crossing in front of the Frog and the resulting Frog. We split the road and river solutions into separate feature sets to further distinguish the innate differences between these two tasks. Thus, if a car was in a box, the road feature would have a value of 1. If a river object or an open home were within one of the Frog’s boxes, the river feature would have a value of 1. If there is no object within a box or a closed home, the feature would have a value of 0. The feature set also included a normalized y-location of the Frog. It is normalized in the range $[0, 1]$ to match the range of the other features. This feature set did not include the ability to determine if the turtles in the game were diving. Thus, we disabled the diving turtles in all of the systems. The diving turtles could be activated again and trained with if we modified the current feature set.




Road			River		
0	1	1	0	0	0
0		0	0		0
0	0	0	0	0	0

Figure 5.1: A possible scenario for the Frog while crossing the road. The figure on the left provides a screenshot of the scenario in the game. The figure on the right provides the resulting feature representation.

5.1.2 Table-Based Q-Learning in Frogger

The first RL agent we wrote was a table-based implementation of the Q-Learning algorithm. As mentioned, this is a “brute force” method of Q-Learning. This implementation took 200,000 episodes to have a reasonable policy, which took over 3 days of training. One episode in Frogger is one run of the game which is terminated by either placing Frogs in all five homes or the Frog dying. This starting point ensured that our feature set sufficiently represented the state-space. We believed, however, that a feature-based representation of the Q-Function would result in a much smaller state space, and thus much faster learning.

5.1.3 Q-Value Approximation

We first attempted to use a linear combination of the features to approximate the Q-value. This meant a Q-value was defined as: $Q(s, a) = \sum_i^n w_i f_i(s, a)$. This approximation did not find a reasonable policy for Frogger with any amount of training. The agent actually learned to avoid entering the road and river to avoid death altogether. The risk of death appeared to be not worth the reward received at the homes. We hypothesize this happened in the linear combination version because the Frogger problem is inherently non-linear. For half the problem, the Frog needs to learn how to avoid the objects moving around it (in the road) and the other half the Frog must collide with the objects moving around it (in the river). We determined we needed to utilize a non-linear approximation of the Q-value, along with our separated feature set, to solve this non-linear problem.

We used a quadratic function for our non-linear representation. This function used the linear combination, like Eq. (3.6), and the product of each feature value to add the non-linear

aspect of the function:

$$Q(s, a) = \sum_{i=0}^n \sum_{j=i}^n w_{i,j} f_i(s, a) f_j(s, a) + \sum_{i=0}^n w_i f_i(s, a) \quad (5.1)$$

This representation found an adequate solution to the road sub-problem but still had some trouble with the river portion of the game. We reviewed our reward structure and determined that the agent needed to have some indication that it was doing something right as it moved closer to the homes. We added a +5 reward for reaching the midway point and +10 for reaching a home. After the agent places a Frog in all five homes, an additionally +25 reward is provided to the agent. Finally, with each successful action taken in the river that moved the Frog between the rows of river objects (when the Frog moved vertically up or down), a $0.1 * \text{rownumber}$ reward was given. This helped the Frog learn a reasonable policy to perform well in both the road and river sub-problems. Once we updated the reward structure, we experimentally found the best hyperparameters to use. We started with hyperparameters near Emigh *et al.*, but changed the final parameters after experimentation [2]. We found that a small ϵ worked best to reinforce the developed weights and not promote unnecessary exploration that typically led to death. We used a small α to discourage the weights from exponentially increasing. Finally, a large γ let the agent focus on rewards far in the future, which makes sense in Frogger, since the rewards will not be immediate upon the agent spawning, but when the agent reached the homes. This then led to the development of an agent with a reasonable policy. Overall, we used the following hyperparameters for our successful agent: $\epsilon = 0.04$, $\alpha = 0.006$, $\gamma = 0.95$.

To prove our results, Figure 5.2 provides the average episode reward of the agent. The episode reward is the sum of all rewards received by the agent in a single episode. To determine the average episode reward, we had the agent perform 30 exploitation runs and recorded the episode reward for each of the 30 runs. An exploitation run is when the agent only exploited its policy and did not update the Q-values. Once the 30 exploitation runs were complete, we took the average of the episode reward across all 30 exploitation runs and returned it as the average episode reward for that given point in training. We paused

training every 100 episodes and determined the average episode reward at that point. We ran 30 separate trials of the given experiment and subsequently averaged the average episode reward across those 30 trials to produce the plot in Figure 5.2. The average episode reward will be used to compare the performance of all agents in the remainder of this report and is defined the same way across all experiments. Figure 5.2's overall average is 24.1. This number qualitatively means the agent, on average, placed 1 Frog in a home and either placed a second Frog in the home or died along the way.

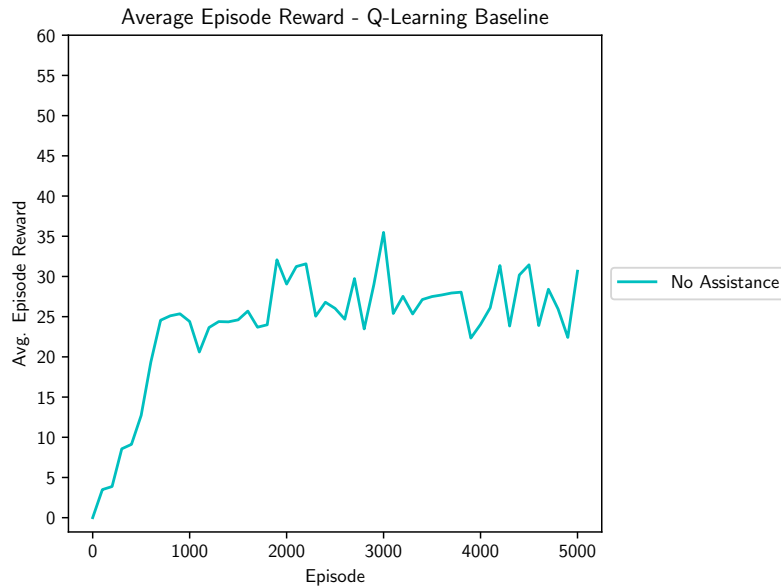


Figure 5.2: Average episode reward of the Q-Learning baseline for Frogger. The average of all of the rewards is 24.1.

5.2 Computer Agent Baselines

In this section, we discuss the three computer agent teaching systems we implemented in Frogger to compare against our human systems. For all of these systems, we used the the set of weights with the highest average episode reward across all 30 trials of the Q-Learning baseline to act as the computer agent teacher.

5.2.1 Advise Important

The Advise Important system, developed by Torrey and Taylor, uses a computer agent that will provide advice in important states [16]. Section 4.2.2 provides the overall explanation of Torrey and Taylor’s system. When an agent is selecting an action and the Importance of a state is over a human-defined threshold, t_t , then the teacher will provide the best action according to the teacher’s developed policy. The importance of a state is defined as: $I(s) = \max_a Q(s, a) - \min_a Q(s, a)$. The last important piece of the system is that the teacher is given a budget which limits the amount of advice the teacher can give throughout training. For every action provided by the teacher, the budget count decreases by one. The algorithm for action selection is:

Algorithm 1 Action Selection-Advise Important

```

procedure ACTIONSELECTION( $s, t_t, n$ )                                ▷ State  $s$ , threshold  $t_t$ , budget  $n$ 
  if  $n > 0$  and  $I_t(s) > t_t$  then
     $n \leftarrow n - 1$ 
    Return advice with teacher policy:  $\pi_t(s)$ 
  else
    Return student action:  $\pi_s(s)$  or exploration (random) action
  end if
end procedure

```

We tried a large number of experiments for this system in order to find the correct budget and threshold needed to have a suitable teacher and student interaction. To evaluate and compare the results, we measured the average episode reward, like we did with the Q-Learning agent alone. Again, with each run of an experiment, training was paused every 100 episodes and we averaged the total reward received over 30 test runs where the student agent used its current policy developed (with no random action). Additionally, we ran 30 separate trials of every experiment to decrease the amount of variance one trial of an experiment may introduce. We ran each experiment on the USNA Cray supercomputer, Grace, to allow for multiple experiments to run in parallel. To measure the use of the teacher, we utilized a statistic used by Amir *et al.* titled cumulative attention [1]. Cumulative attention is the average of the number of states in which the teacher was asked to provide advice within a

Budget:		3000	5000	20000	50000
Threshold	10.0		30.2 ^(a)	32.0 ^(b)	42.9 ^(c)
	11.0	24.0 ⁽ⁱ⁾	22.3 ^(g)		
	12.0	22.6 ^(h)	40.0 ^(d)	39.1 ^(e)	38.9 ^(f)
	13.0	32.7 ^(j)	35.6 ^(k)		
	14.0	35.3 ^(l)			

Table 5.1: Experiments for the Advise Important system. The experiment numbers are given in the cell of the table, with the average reward across the average of the 30 runs.

100 episode window. Each 100 episode frame’s value is added to the previous 100 episode frame to track the ratio cumulatively. We decided to slightly modify this statistic by taking the ratio of the number of states the teacher provided advice to the total number of states the agent observed. This would allow the different experiments to be directly compared, as some experiments would have more states actually seen within the window, which skews the statistic overall. A ratio will allow this to be compared directly to see the number of states the teacher had to advise in to the actual number of states seen. Additionally, since this a cumulative statistic, each 100 episode attention measurement is added to the last 100 episode’s measurement. Thus, on a cumulative attention graph, the y-axis value at a given point is the combined ratio at that point in training. When the plot flattens out, that means that the budget has ran out and the ratio will stay constant for the remainder of training. Again, the cumulative attention for a given experiment is the average cumulative attention over 30 trials of the experiment.

Table 5.1 provides all of the experiments we ran. The average rewards over 30 runs is given for each experiment. Each experiment is also labeled with its corresponding label.

Figure 5.3 provides all of the experiments’ average episode reward on one plot. Figure 5.4 provides the three best performing experiments (experiment *c,d,e*) on one plot. These experiments had the highest average episode reward. Figure 5.5(a,b,c) provides the individual episode rewards for the three best experiments. Figure 5.6 provides the cumulative attention of all 12 experiments.

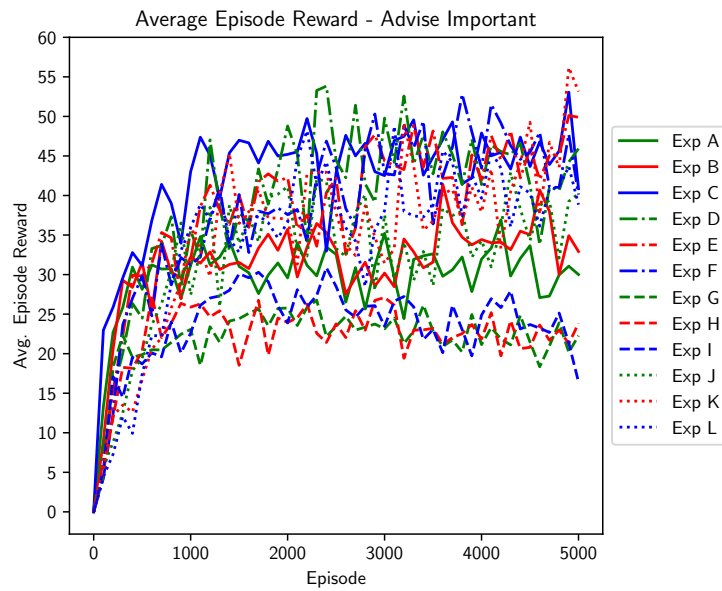


Figure 5.3: Average episode reward for each of the 12 experiments.

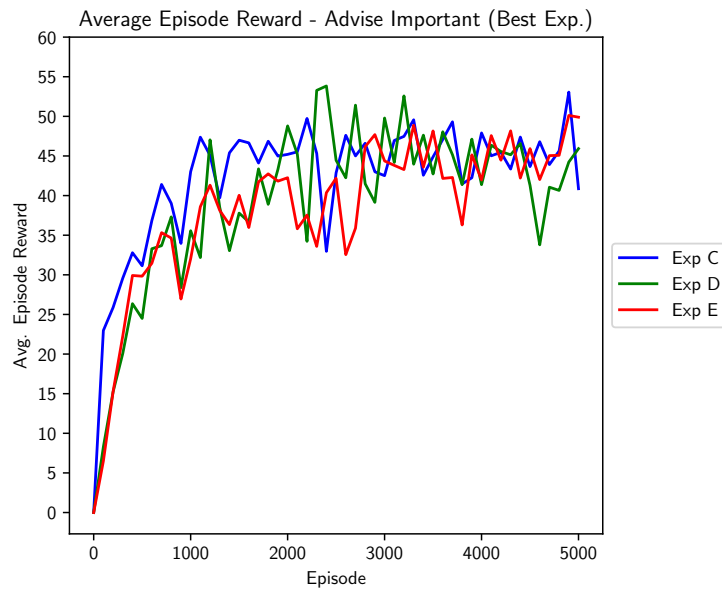
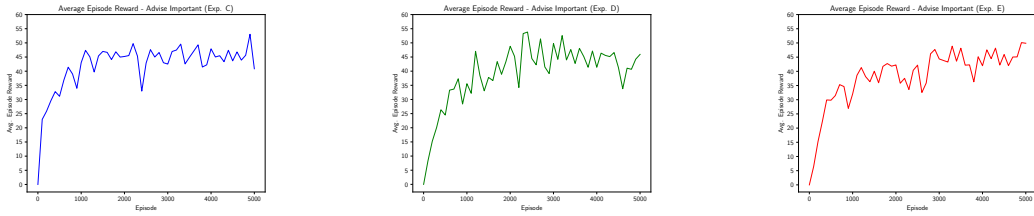


Figure 5.4: The three best experiment's average episode reward on one plot.



(a) The average episode reward of experiment *c*. The average of this plot is 42.9.

(b) The average episode reward of experiment *d*. The average of this plot is 40.0.

(c) The average episode reward of experiment *e*. The average of this plot is 39.1.

Figure 5.5: The individual average episode reward of the top performing experiments. Experiments *c*, *d*, and *e* had the highest average episode reward across all of their average episode rewards.

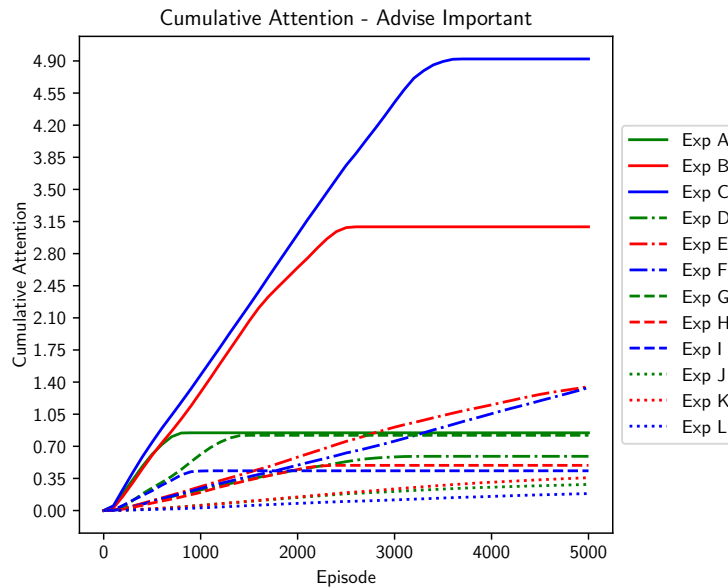


Figure 5.6: A plot of each experiment's cumulative attention.

These show that the Advise Important system worked in Frogger similarly to how it worked in Torrey and Taylor's original development of the system in Pac-Man. As compared to the strictly Q-Learning agent (with no teacher), all of these systems developed a reasonable policy with much less training time. Additionally, the amount of reward (average) for each individual experiment was either as good or better than the Q-Learning baseline.

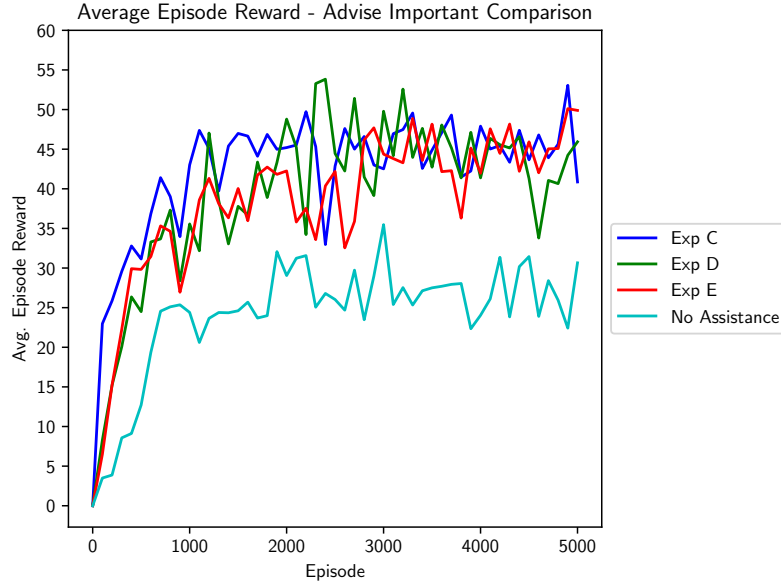


Figure 5.7: A plot comparing the three best Advise Important experiments (c , d , and e) and the Q-Learning agent alone.

We compared the best performing experiments (c , d , and e) to the Q-Learning baseline in Figure 5.7. It is important to note, however, the drastic differences between the cumulative attention of each of the experiments. Experiment c , the experiment with the highest average reward, had a very large ratio of number of advice states to total states seen. By reaching 4.92, it shows that the teacher consistently provided a lot of advice to the agent. This makes sense, as this experiment had a relatively low threshold and a very high budget, which meant the teacher gave advice more frequently and for a much longer time than the other experiments. Ideally, we want an agent that performs as well as experiment c , with a lower cumulative attention curve, as this means the teacher provides meaningful advice in the most pivotal states. With that goal, experiment d appeared to be the “best” parameters for this system. Experiment d still had a high reward average, along with a low cumulative attention curve. With an average of 40.0, this means the agent placed 2 Frogs in the home on average and either placed a third Frog in the home or died along the way. Experiment d also ran out of budget at roughly the 2800 episode mark. Therefore, we chose to use those parameters for future experiments.

Regardless, we have proven that the computer teacher system Advise Important provides a better policy at a faster rate than the Q-Learning agent alone.

5.2.2 Mistake Correcting

Another one of Torrey and Taylor’s computer agent systems is the Mistake Correcting system [16]. With Mistake Correcting, the teacher provides action advice when the state the student is in has an importance greater than the human defined threshold t_t (importance measured the same as in Advise Important), the teacher hasn’t used the entire advice budget, **and** the action the student would choose in the state is different than the teacher’s action it would provide. This system should be able to produce similar results as the Advise Important system, but with the teacher providing far less advice than those systems (a lower cumulative attention curve). Algorithm 2 provides the algorithm for Mistake Correcting.

Algorithm 2 Action Selection-Mistake Correcting

```

procedure ACTIONSELECTION( $s, t, n$ )                                ▷ State  $s$ , threshold  $t$ , budget  $n$ 
   $a_{student} \leftarrow \pi_s(s)$  or exploration (random) action
  if  $n > 0$  and  $I(s) > t$  and  $\pi_t(s) \neq a_{student}$  then
     $n \leftarrow n - 1$ 
    Return advice with teacher policy:  $\pi_t(s)$ 
  else
    Return student action:  $a_{student}$ 
  end if
end procedure

```

For Mistake Correcting, since we had a better idea of the best parameters for the threshold and budget, we ran two experiments that both produced excellent results. We tracked the same information as we did with Advise Important. We ran each experiment 30 times, where the cumulative attention was tracked every 100 episodes and we paused training every 100 episodes to run 30 test runs to get an average episode reward every 100 episodes. Figure 5.8 provides the average episode reward for both of the experiments. Figure 5.9 provides the cumulative attention for both experiments. Experiment *a* used a threshold of 12.0 and a budget of 5000, while experiment *b* used a threshold of 12.0 and a budget of 3000.

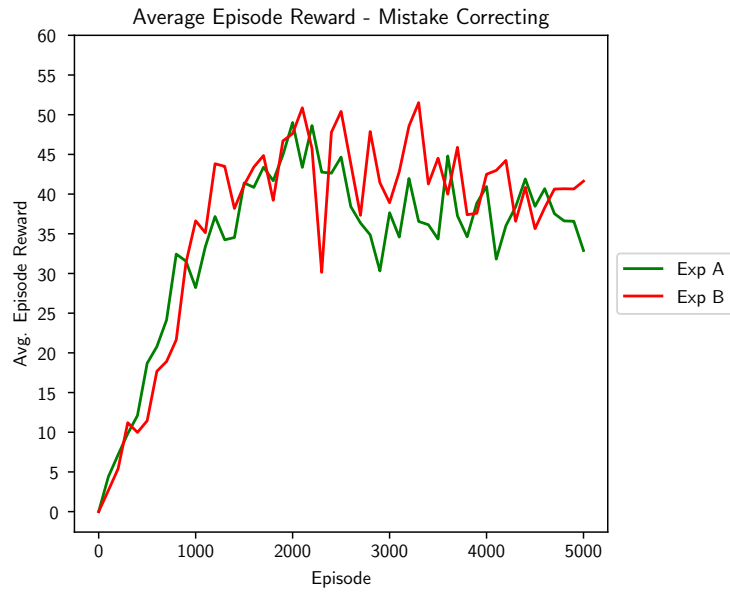


Figure 5.8: Average episode reward for each experiment. Experiment *a* used a threshold of 12.0 and a budget of 5000. Experiment *b* used a threshold of 12.0 and a budget of 3000.

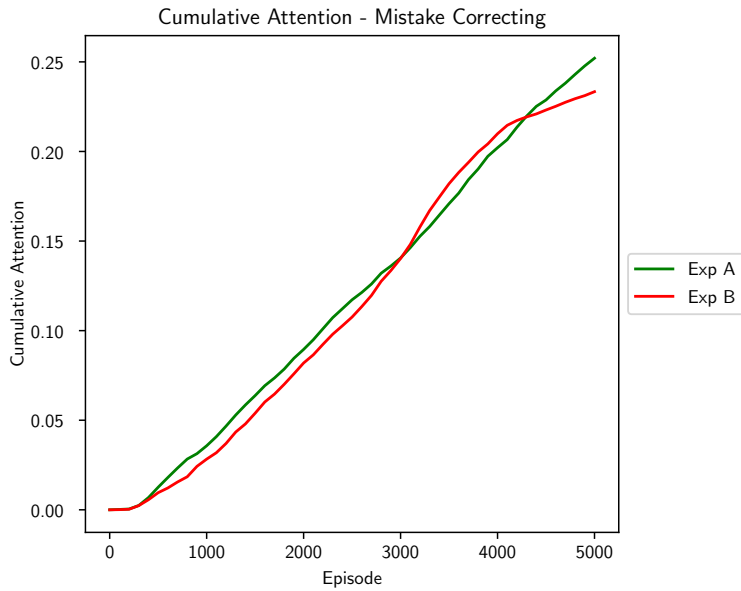
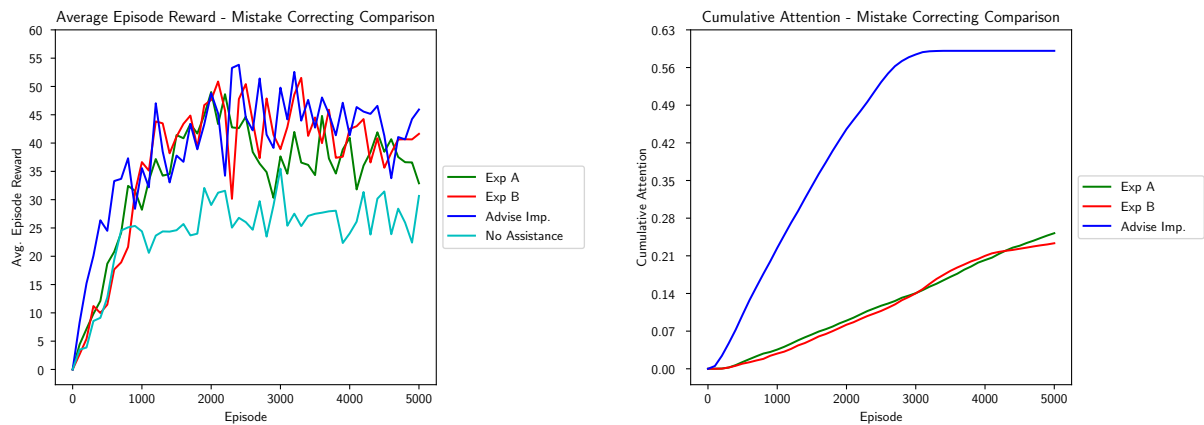


Figure 5.9: A plot of each experiment's cumulative attention. Experiment *a* has a threshold of 12.0 and budget of 5000, while experiment *b* has a threshold of 12.0 and a budget of 3000.

The average reward for experiment *a* was 34.8, while the average for experiment *b* was 37.1. Both of these averages mean that the agent placed 2 Frogs in the home, on average, and either placed a third Frog in the home or died along the way. Figure 5.10(a) provides a comparison of the average episode reward for the Q-Learning baseline, the best Advise Important experiment (experiment *d*) and both Mistake Correcting experiments. Figure 5.10(b) provides the cumulative attention curves for the best Advise Important experiment (experiment *d*) and both Mistake Correcting experiments.



(a) Comparison of the average episode reward.

(b) Comparison of the Cumulative Attention.

Figure 5.10: The comparison of the Q-Learning baseline, the best Advise Important experiment (*d*) and both Mistake Correcting experiments. (a) provides the comparison of average episode reward and (b) provides the comparison of the cumulative attention curves.

Notice that both of these experiments produced similar average episode reward plots and values to Advise Important, but with a dramatically smaller cumulative attention than even the best Advise Important system. Again, these were the expected results, as there may be a large number of states that the student happens to agree on the action that the teacher would have produced with its advice. We showed that Torrey and Taylor’s Mistake Correcting system not only performed better than their Advise Important system, but also resulted in a better policy at a faster rate than the Q-Learning agent alone.

5.2.3 Ask Important-Correct Important

The last baseline system we tested was Amir *et al.*'s best system, Ask Important-Correct Important. This system first had the student evaluate the importance of the state according to the student's Q-values ($I_s(s)$) and student threshold t_s . If that value was over the student threshold, then the Mistake Correcting system in Torrey and Taylor was used, by checking if the importance of the state ($I_t(s)$) was above the teacher threshold t_t , the budget had not ran out, and the action the student would take was not equal to the teacher's provided action. Ideally, the policy created performs better than the two Torrey and Taylor systems by having a higher average episode reward and lower cumulative attention curve. The Ask Important-Correct Important algorithm is:

Algorithm 3 Action Selection-Ask Important-Correct Important

```

procedure ACTIONSELECTION( $s, t_t, t_s, n$ )           ▷ State  $s$ , thresholds  $t_t, t_s$ , budget  $n$ 
   $a_{student} \leftarrow \pi_s(s)$  or exploration (random) action
  if  $I_s(s) > t_s$  then
    if  $n > 0$  and  $I_t(s) > t_t$  and  $\pi_t(s) \neq a_{student}$  then
       $n \leftarrow n - 1$ 
      Return advice with teacher policy:  $\pi_t(s)$ 
    else
      Return student action:  $a_{student}$ 
    end if
  else
    Return student action:  $a_{student}$ 
  end if
end procedure

```

We ran 4 experiments for Ask Important-Correct Important. Table 5.2 provides the experiment numbers and parameters used, along with the overall average reward. We tracked the same statistics (average episode reward every 100 episodes and cumulative attention averaged over 30 separate runs for each experiment) for this system. Figure 5.11 provides the average episode reward of all of the experiments and Figure 5.12 provides the cumulative attention for all four experiments.

Experiment	Teacher Threshold (t_t)	Budget	Student Threshold (t_s)	Average Reward
<i>a</i>	12.0	5000	9.0	36.37
<i>b</i>	12.0	5000	10.0	36.95
<i>c</i>	12.0	3000	9.0	36.70
<i>d</i>	12.0	3000	10.0	33.95

Table 5.2: Experiments for the Ask Important-Correct Important system. Each row provides the experiment label, parameters used for the experiment, and average overall reward.

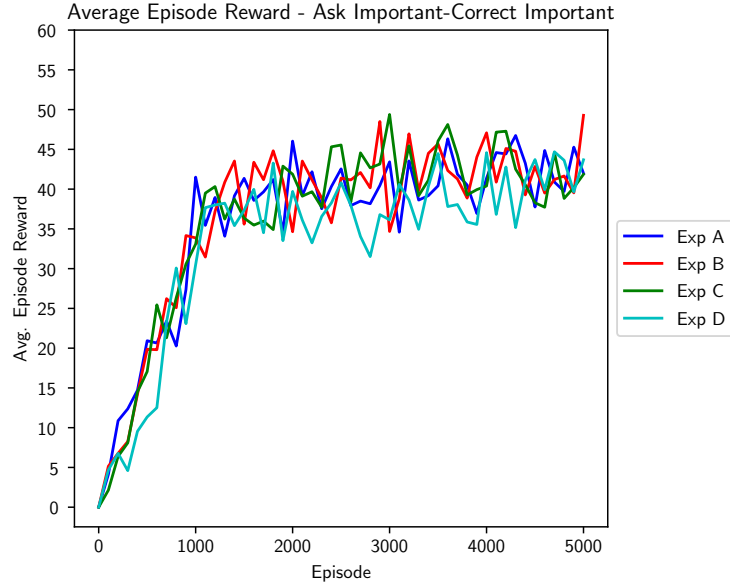


Figure 5.11: Average episode reward for each Ask Important-Correct Important experiment. Table 5.2 provides the parameters used for all four experiments.

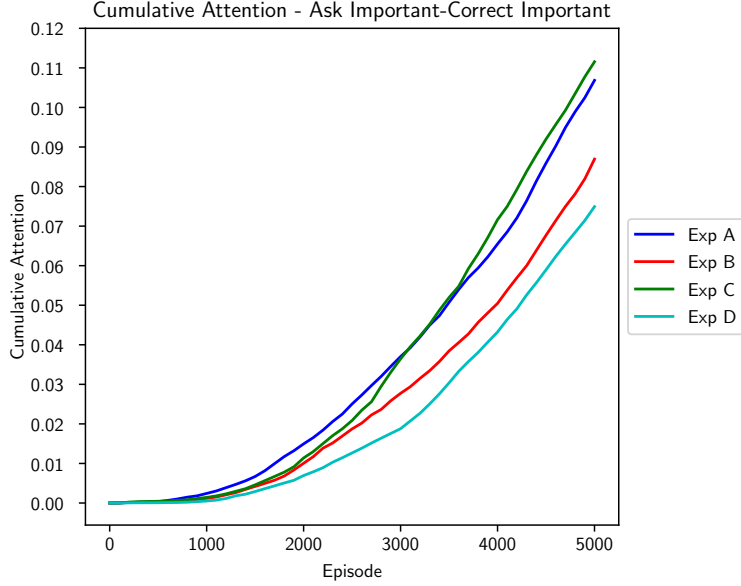
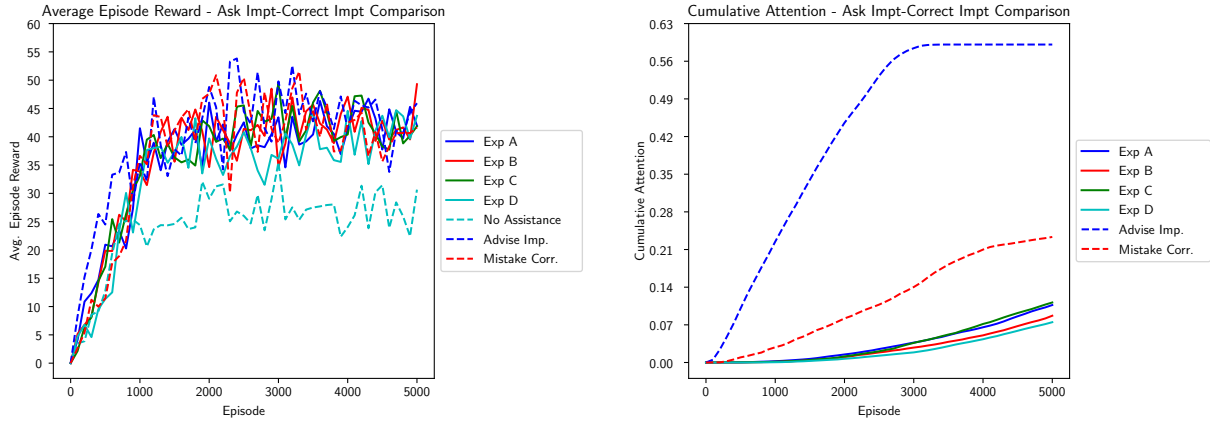


Figure 5.12: A plot of each Ask Important-Correct Important experiment’s cumulative attention. Table 5.2 provides the parameters used for each experiment.

These results show that the Ask Important-Correct Important systems do in fact produce a better policy quicker than a Q-Learning agent alone. Again, an average near 37.0 signifies the agent typically placed 2 Frogs in the home and either placed a third Frog or died along the way. The systems did not necessarily produce higher reward averages than the other computer teaching systems, however, this system, as Amir *et al.* showed, had a much lower cumulative attention curve, meaning the teacher was not utilized nearly as much as the other systems. Figure 5.13(a) provides a comparison of the average episode reward of each Ask Important-Correct Important experiment, the Q-Learning baseline, and the best experiments from Advise Important (*d*) and Mistake Correcting (*b*). Figure 5.13(b) provides the comparison of cumulative attention of all Ask Important-Correct Important experiments and the best experiments from Advise Important (*d*) and Mistake Correcting (*b*). An interesting observation for the Ask Important-Correct Important system is that the teacher did not really provide any advice within the first 1000 episodes and provided the majority of its advice near the end of training. Regardless, this system still proved the

abilities of a computer teacher system and its success in Frogger.



(a) Comparison of the average episode reward.

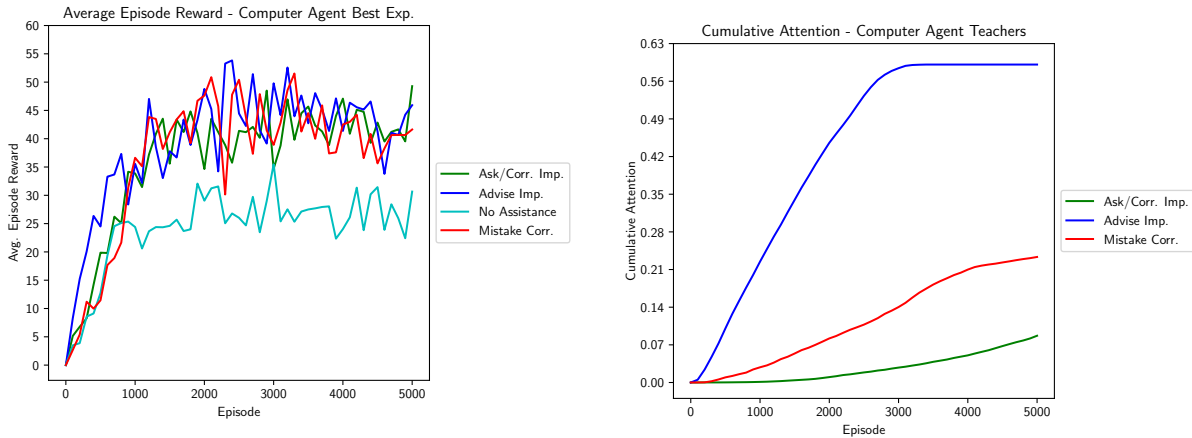
(b) Comparison of the Cumulative Attention.

Figure 5.13: The comparison of the Q-Learning baseline, the best Advise Important experiment (*d*), the best Mistake Correcting experiment (*b*) and all four Ask Important-Correct Important experiments. (a) provides the comparison of average episode reward and (b) provides the comparison of the cumulative attention curves.

5.3 Comparison of Computer Systems

After having implemented all previous work, it is evident that the computer agent teachers provided tremendous assistance to the learning agent in training. Each system enabled the agent to produce a better overall policy much faster than a Q-Learning agent alone. As previously mentioned, it still is desirable to have a human expert as a teacher, as there is no reason to train another agent after already having one with an optimal policy. Additionally, in complex domains like Frogger, a human expert may help solve the planning requirement needed to produce an even better performing agent. Before discussing the human systems developed for this project, we present the best experiments for each of the computer agent teaching systems, plotted on the same plot, in comparison to the standard Q-Learning agent. Figure 5.14(a) provides the average episode reward of the Q-Learning agent alone, Experiment *d* of the Advise Important system (threshold of 12.0, budget of 5000), Experiment *b* of

the Mistake Correcting system (threshold of 12.0, budget of 3000), and experiment *b* of the Ask Important-Correct Important system (student threshold of 10.0, teacher threshold of 12.0, budget of 5000). Figure 5.14(b) provides the same experiment's cumulative attention plots.



(a) The average episode reward of each system's best experiment and the Q-Learning agent alone.

(b) The cumulative attention for each system's best experiment.

Figure 5.14: Comparison of each system's best experiment by plotting the average episode reward and cumulative attention on one plot. Advise Important's best experiment was *d*, Mistake Correcting's best experiment was *b*, and Ask Important-Correct Important's best experiment was *b*.

Chapter 6

Human Teacher System - Time Warp

In this chapter, we introduce our first human expert teaching system: Time Warp. We first will introduce the concept of Time Warp and the design of the system. We then report the results of the Time Warp systems after bringing human teachers in to interact with the system.

6.1 Time Warp Development

After reviewing the multitude of human expert and computer agent teaching systems, the need for a human-teacher based system that was as effective as the computer-teacher systems. We wanted to create a system that accomplished the following goals:

- Produce an optimal policy faster than a Q-Learning agent alone
- Develop an optimal policy that had a better average than a Q-Learning agent alone
- Mimic average episode reward and cumulative attention plots for computer agent teaching systems
- Solve the planning requirement evident in more complex domains than previous teaching systems used
- Allow for the agent to train without the presence of a human

To begin achieving these goals, we hypothesized that a system that allowed the human to assist in the exploration of the space would have a better effect than allowing the human to control a feedback signal. Thomaz and Breazeal’s research showed that humans are more inclined to provide guidance, rather than feedback [15]. This means that we needed to find a way to allow the human to provide exploration steps to the agent. We first developed the idea of replicating Torrey and Taylor’s advise important system with a human as the teacher [16]. This meant that when the human believed the agent was in an important state (according to the human’s definition of importance, not a mathematical representation of importance), the human would freeze training and provide a single action to the agent. We actually developed this system, but determined that a human would not be able to react fast enough to signal actions in important states, as an agent moves much faster than a typical human. Additionally, we found that this system would not allow the human to control full exploration of the space as it would require the human to provide a large number of advice actions just to move the agent in one episode. Humans do not have nearly as much patience as a computer teaching system, so we had to take that into account for our system design.

To reduce the amount of times the human had to provide actions and to allow the human to properly provide exploration, we decided to implement a system we titled “Time Warp.” This system would have the agent training normally (a Q-Learning agent alone), but if the human notices the agent went past an important state, the human could freeze training and reverse the states the agent had visited in a given time frame. Once the human determined the state they wanted to resume training from, the human would have full control of the agent. This meant the human would provide a sequence of actions to help the agent explore the state space. Once the human had control, every action the human took with the keyboard would be provided as the action the agent would take and what action was used to update the Q-values. In between these directly inputted actions from the human, NOOP actions would be provided as the advice from the human. NOOP actions, for the agent, are simply to do nothing and stay in the current position. All of these actions were provided to the Q-Learning algorithm to update the weights of the agent. This system is not explicitly learning from demonstration, another common RL technique, since the human would be in charge of

inherently assisting the agent in better exploration of the space by only providing actions in important states witnessed during training.

When the human teacher decided to activate the “Time Warp” system, we calculated the amount of states the human could reverse time. If the human activated the system within 50 states after a death, the human could reverse time from the current state up to 50 states before the death. If the human gave the signal after 50 states have passed since the last death, the human could reverse time from the current state back to 50 states before the signal was given. This allowed the human to strategically find the important state they wanted to begin training at and then provide a sequence of exploration steps following that important state. With this amount of freedom, the human could choose the strategy they wished to employ. They could simply play Frogger by always having control, they could provide a single action in one state, or utilize a strategy somewhere in the middle of these two extremes. These strategies were tracked by comparing how many episodes the human trained with their agent. If the human only trained for a small number of episodes, they used their budget quickly, which meant they took control of the agent often. The humans that trained for a longer time used their budget more sparingly and thus only provided a small number of actions each time they took control.

Once the human finished, either by using their budget of 5000 actions, quitting on their own, or running out of the ten minute time limit provided, the weights developed with the human trainer were saved. From that point, we would complete the training by running a standard Q-Learning agent from the last episode the human completed until 5000 training episodes using the last developed weights the human trainer’s agent used. This would allow the exploration assistance provided by the human to be utilized and continued for a regular agent to train. The human’s exploration and training would directly influence the actions taken by the Q-Learning agent and help develop a better policy. This is an interesting divergence from the computer systems, as most other systems were given a large enough budget to not run out within 100 episodes, but humans easily could run out of their budget this quickly. Our systems rely heavily on early advice, even with humans that spend their budgets conservatively.

6.2 Experimental Setup

To test our Time Warp system, we used 49 subjects interested in participating in the experiment. The subjects were given a very brief explanation of how RL worked and how to interact with the system. Essentially, subjects were told that the agent will receive a feedback signal for doing good and bad actions. We did not provide the exact reward structure of Frogger, but rather told the subjects that doing well at Frogger will mean the agent will receive positive feedback, while dying will provide negative feedback. The subjects were given a demo on how to interact with the system and the various controls needed. No explicit strategy was given to any of the subjects, other than that there were two extremes: essentially playing Frogger or giving one action at a time and returning control back to the agent. Appendix A provides a script of what we told the subjects prior to their interaction with the system.

Similarly to the computer agent baselines we tested, we tracked average episode reward and cumulative attention. For average episode reward, we paused training every 100 episodes and ran 30 test runs (by exploiting policy and not updating Q-values). That average would be returned as the average episode reward for that point in training. We tracked cumulative attention the same, by recording the ratio of states the human provided advice to the number of states seen within a 100 episode frame and adding to the last 100 episode frame. For every human’s interaction with the system, we continued training from the episode the human stopped training at (with the weights developed at that point) until 5000 episodes with a standard Q-Learning agent. When we continued training, we ran 30 separate trials of standard Q-Learning for each human trainer’s developed weights after the human finished training. This was to ensure we did not have variance between training runs of the standard Q-Learning agent. For humans that participated in more than 100 episodes, we saved the weights for every 100 episodes and ran 30 test runs with the human’s developed weights. This would become the average episode reward for all 30 trials at that particular episode. The cumulative attention plots will all flatten out after the human finishes training, as the numerator of the ratio for every follow-on 100 episode frame will be 0, since the human

will no longer be providing advice. This means that although the human’s ratio may be high compared to other computer agent systems, our system will show more stability and no change after a certain episode.

6.3 Experimental Results and Discussion

Following the 49 subjects’ interaction with the system and finishing the training we continued following the human training, we found positive results. Figure 6.1 provides the average episode reward averaged across all 49 experiments compared to the Q-Learning agent alone as developed in Chapter 5. The overall average was 38.48. This average means the agent placed 2 Frogs in the home on average and typically was able to get 3 Frogs in the home and died any other time.

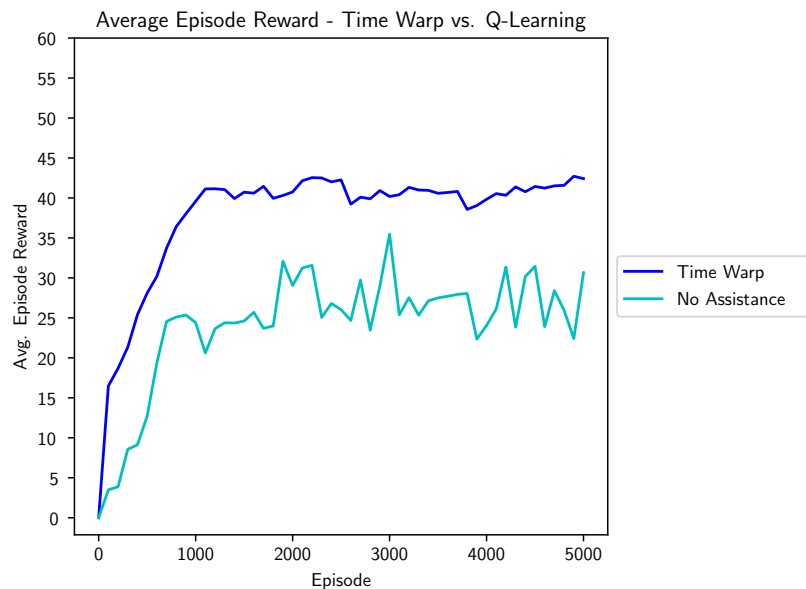


Figure 6.1: The average episode reward averaged over all 49 human Time Warp trials. This is compared to the Q-Learning agent alone.

Interestingly, this graph provides more stability in later episodes than most of the computer agents, despite having a slightly lower overall average. Additionally, it is evident that

a reasonable policy is developed faster than the Q-Learning agent alone, which is one of the primary goals of this system. Not only did we prove that our system was better than the Q-Learning agent alone, but also competitive with the other computer agent systems. Figure 6.2 provides a comparison of Time Warp’s average episode rewards to the best performing computer agent systems.



Figure 6.2: A plot comparing the Time Warp system and the best computer agent teaching system experiments.

Figure 6.2 shows that the Time Warp system actually developed a policy faster than any of the best performing computer agent teaching systems. Although the average was slightly lower than some other computer teaching systems, it also provides a more stable plot and a reasonable policy at a faster rate.

Another important consideration for evaluating our system is the average cumulative attention. We took the average cumulative attention across all 49 experiments. We compare the cumulative attention of the Time Warp system with the best experiments from the computer agent teaching systems in Figure 6.3.

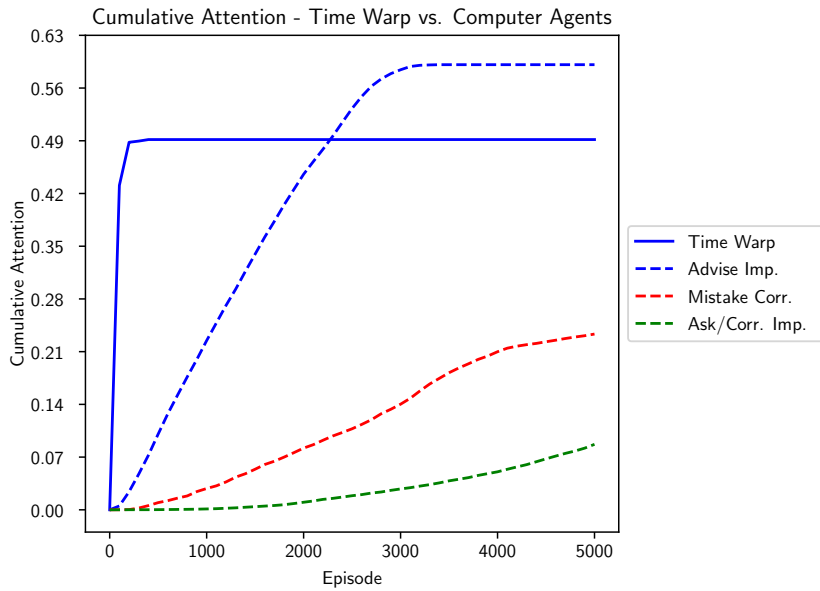


Figure 6.3: A plot comparing the cumulative attention of Time Warp and the best performing computer agent teaching systems.

Although our cumulative attention graph has a higher value than Mistake Correcting and Ask Important-Correct Important, we still had less cumulative attention than the Advise Important system, which we used to base our system’s design. Additionally, this makes sense, since most of the humans used their budget within the first 100 episodes of training, driving the cumulative attention up substantially within the 100 episode frame. This plot also proves that front-loading the training with the human still worked. While the other systems had to have a teacher present for most of the training of the agent, our system only needed human interaction for a short period of time and still produced similar results.

Another important statistic we gathered from our data was the difference between early and late trainers. We first took the average episode that the human trainers stopped training, either because they quit on their own, the ten minute time limit ran out, or their budget ran out. The average episode that humans stopped training was episode 96. We used this average to separate the early and late trainers. If an individual trainer stopped training before or at the average episode, they were denoted as an “early” trainer. Conversely,

trainers that trained past the average episode stopping point were denoted “late” trainers. We then averaged the episode reward for each groups and plotted both, as seen in Figure 6.4.

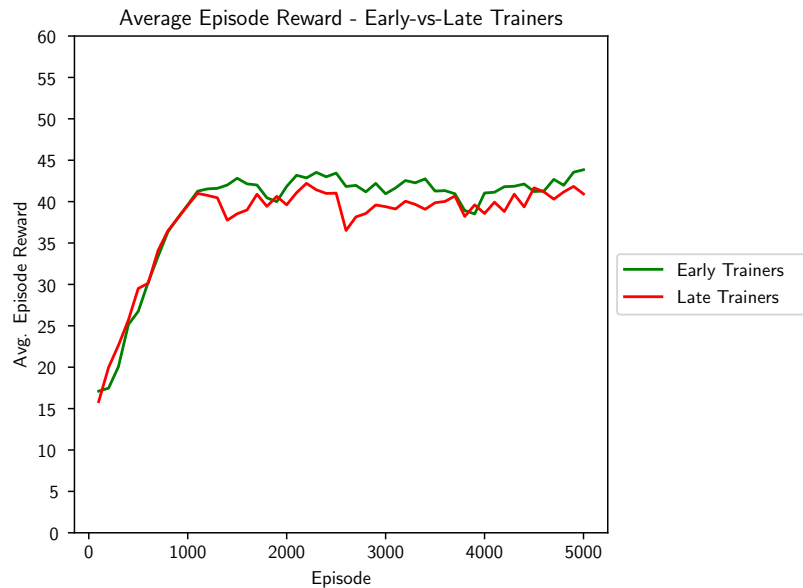


Figure 6.4: A comparison of the early and late trainers’ average episode reward. Early trainers were the human experts that trained for less than 96 total episodes and late trainers were those that trained more than 96 episodes. 96 episodes was the average stopping episode of all trainers.

Interestingly, the early trainers had a better overall average episode reward of 39.1, while the late trainer’s average was 37.8. We originally hypothesized that the late trainers would have a better average, as they had more time to provide advice to the agent. However, as Figure 6.4 shows, the early trainers ended up having a better overall average. The difference is small, which shows that it really does not make a significant effect on the overall performance of the agent, but it is a interesting insight to the training time taken by each human expert. Regardless, both of these averages indicate the agent is placing at least 2 Frogs in the home and either places a third Frog in the home or dies along the way.

Another statistic that we gathered from the human expert data was the difference between “good” and “bad” trainers. To determine which group a given trainer would be placed in,

we took the weights the human had developed at the conclusion of their training and ran 30 runs of exploiting the policy at that given point. We collected the average across these 30 runs and compared against all trainers. The average for these 30 runs was 11.7, thus if a trainer had an average higher than 11.7, they were considered a “good” trainer, while those under the average were “bad” trainers. Figure 5.5 provides a plot of the average episode reward for both groups (averaged across all trainer’s rewards).



Figure 6.5: A comparison of the good and bad trainers’ average episode reward. Bad trainers were those who had an average episode reward immediately after teacher training (utilizing the weights developed when the trainer finished training) less than the entire groups overall average of 11.7. Good trainers were those that had an average episode reward immediately after teacher training that was greater than this average.

Another surprising insight; the “bad” trainers actually had a better average than the “good” trainers. The bad trainers had an overall average of 39.5, while the good trainers had an overall of 37.1. Again, both of these averages indicate the agent places at least two Frogs in the home on an average run. Initially, we hypothesized that the good trainers would actually provide a better overall average in the end, but clearly the skill of the trainer at the time the trainer quit did not matter in the overall performance of the agent. This also

indicates that we do not necessarily have to have experts that completely understand RL or the domain itself. Even those that were bad at training an agent still produced an agent that had a reasonable policy to perform well in Frogger.

Finally, another interesting insight in the data was the separation of two groups when viewing all trainer's individual average episode reward. Figure 6.6 provides a plot of all 49 trainer's average episode reward.

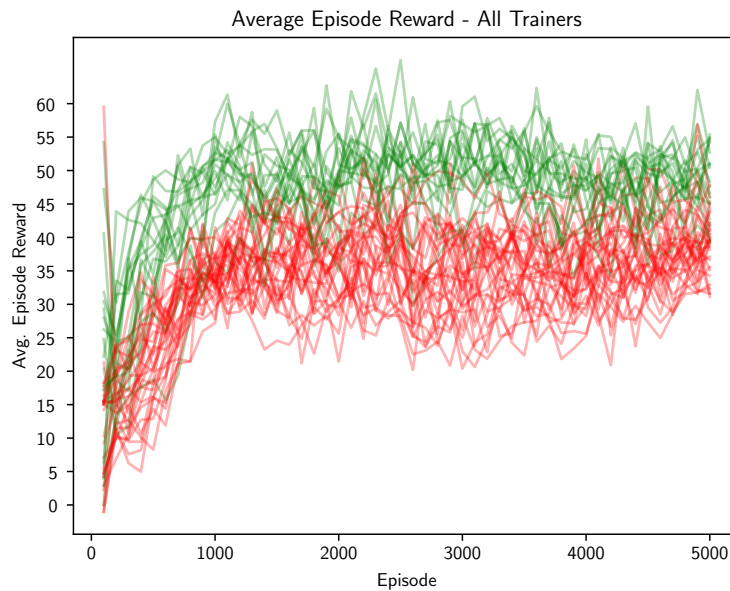


Figure 6.6: A plot of every trainer's average episode reward. Notice the separation of two groups within the plot, highlighted in red and green. Effectively, this plot shows the true good and bad trainers. The trainers that fell in the top group utilized a better strategy that increased exploration of the space, which allowed for a better policy to be developed.

We highlighted two distinct groups within the plot. Interestingly enough, these plots did not split evenly as good versus bad trainers (at the conclusion of human training) or early versus late trainers. Little crossover between the groups showed that once a certain trainer was in a group, they tended to stay in their group. There is no inherent strategy or technique that correlated to a trainer having their fully developed agent being placed in either of these groups. Most likely, the trainers that fell in the top group had a better strategy that ended up providing the agent with better exploration at the beginning of training, which allowed

the follow-on training to develop a better policy. With more investigation, we could find a correlating variable that determines whether or not a trainer gets placed in these two distinctive groups.

Chapter 7

Human Teacher System - Curriculum Development

In this chapter, we introduce the second human expert teaching system: Curriculum Development. We will first introduce the concept and design of the Curriculum Development system. We then will report the results of the Curriculum Development systems after allowing human teachers to interact with the system.

7.1 Curriculum Development Design

While the Time Warp systems allowed for direct control of the agent's exploration on the human's part, we also designed a system that presented a unique way of creating exploration. Similar to Maclin and Shavlik's system, we wanted users to be able to define scenarios that they believed were important for the agent to learn [7]. However, we did not want to introduce the use of a programming or scripting language for the user to have to learn, but rather allow direct design of scenarios that the agent should know. Additionally, we wanted to avoid any human advice in this system, as Time Warp had already proven that human advice does in fact work well. Instead, we present a system that allows the user to define scenarios using a simple interface, which then will place the agent in the situation the human designs. From there, the human will launch training and the agent will perform standard Q-Learning until

death. Once the agent dies, the human teacher has another opportunity to create a new situation and thus provide more exploration of the state space. While this system is similar to Narvekar *et al.*, it is different in a number of ways. In particular, their systems create the curriculum during the training of the agent, along the training trajectory. Their systems will modify the training of the agent to train the agent in the necessary subskills, hence the name Curriculum Learning. The agent learns the best curriculum to follow and executes that curriculum during training. Our system actually takes place after an agent has trained for a given time. Then, the curriculum is developed by the human through presentation of important scenarios for the agent to learn. Additionally, the largest difference is that our system uses a human expert to develop the curriculum as opposed to a computer-generated curriculum.

This leads to the title given to the system: Curriculum Development. The human training is providing a curriculum for the agent to learn necessary subskills of the overall Frogger problem. The human defines scenarios that they believe are pivotal for the agent to learn. Then, the agent will learn how to successfully solve those situations with a standard Q-Learning algorithm. By utilizing the same agent for all scenarios, the agent effectively is placed through a curriculum designed by the human expert. This curriculum steps through each individual problem presented by the human teacher and eventually solves the entire problem. We hypothesized that this system will in fact solve the inherent planning requirement needed for Frogger. Frogger requires some amount of planning, as one different move in the river could change the final home the Frog travels to. With Curriculum Development, the human expert can show which homes should be aimed for first or what moves are required to solve the problem as a whole. Although the human is not providing the exact actions to take in these situations, the human can provide situations that the Frog can learn to develop weights as to which homes to aim for first. The Frog will then learn the correct moves on its own with a standard algorithm. By solving the planning requirement, we hypothesize this system will be able to produce a better policy than the policy developed at the point human training began.

The interface designed for the system was not a very complicated setup. Before every

curriculum round, the human teacher was presented with the starting state of the Frogger game. From there, humans would be able to select individual rows of cars, turtles, and logs. The cars, turtles, and logs could then all be shifted to any position the human wanted. This was done by using the left and right arrow keys to move the objects to their desired position. The human could also select where the Frog would start in training by clicking to place the Frog. The Frog would only be placed in valid locations throughout the space. This means in the road zone, the Frog would snap to the only positions possible for the Frog to ever find. In the river, there was more variety, as the Frog could be placed at any horizontal location on one of the river objects. Regardless, the human could never place the Frog in a state that it would die immediately upon the start of training. Once the human was satisfied with the scenario they set up, they would press the space bar and the agent would enter standard Q-Learning until death. Once the agent died, the last scenario created by the human would be loaded up and the human could create another scenario. The human would be finished training after a ten-minute time limit or whenever they wanted to quit training. Additionally, the agent began human training with weights that had been developed after 5000 episodes of standard training. In fact, the weights used were the same weights used by the computer agent teachers in the replication phase of the project. This would allow the agent to have some prior knowledge of a solution and the human will be able to direct the Frogger through the necessary planning requirements to eventually build a better policy for the agent. These weights already performed incredibly well with a standard agent, but ideally the human training would help take these weights and develop them to create a better policy.

An interesting point about this creation is worth noting. The human teacher could never place the Frog in invalid locations such as a death state or location it could never reach in training. However, the human could position the cars, turtles, and logs in relative positions to one another that would never occur in a normal play of the game. This fact allows the agent to learn a more generalized version of Frogger as opposed to eventually finding an exact path (or sequence of actions) needed for every episode. A more generalized solution will allow the agent to fully understand the game of Frogger and perform more like a human

player would perform. This is exactly what Curriculum Development is supposed to do as well. The human expert presents a number of scenarios that it believes are important to learn for an agent. The agent learns these subskills using a Q-value function approximation and a feature set so once the human finishes training, it is able to generalize that training to the entire problem and eventually find a solution to the problem.

7.2 Experimental Setup

For this system, we used a second group of human trainers. We did not want the same trainers using both systems as that could lead to confusion and misinterpretation of each system’s objectives. We invited 15 new trainers to test the system. Each trainer was given a quick demo on how to use the controls of the system. Then, each trainer was told they were supposed to create situations that they thought were important for the Frog agent to know. We instructed them to act as if they were instructing a child who had never played Frogger on what situations were important for the child to learn. The full script is available in Appendix A. Each human trainer was then given the ten minute time limit to construct as many scenarios as they could. Alternatively, the human trainers could quit at any moment when they believed they had presented every situation they could create. We saved every state the human trainers created, the rewards received with each agent’s run of the system (once the scenario had been set), and the final weights developed by the agent at the conclusion of training.

Following the human training, we took every agent’s learned weights and ran 30 separate trials that continued training where the human left off. This continuation of training began at the episode number the human’s training ended. The same standard Q-Learning algorithm was ran for these trials. We tracked average episode reward the same as we did for all other systems by pausing training every 100 episodes and running 30 test runs and took the average reward received from those runs. We did not track cumulative attention, as that statistic does not make sense to track for this system. Rather, we tracked how many total scenarios the human trainer developed throughout training. The next section will present

our results of the Curriculum Development system.

7.3 Experimental Results and Discussion

Following the continuation of training, we found interesting results. Our hypothesis did not turn out to be correct for Curriculum Development. The agent actually did not create a better policy after the human interacted with the system in many instances, but in others created much better policies. We first continued training with a standard Q-Learning agent for 2000 episodes. We kept the same hyperparameters, including the epsilon, used throughout all systems in this report. Figure 7.1 provides the results from the 2000 episode continuation run.



Figure 7.1: The average episode reward averaged over all 15 human trials after training for an additional 2000 episodes. Notice the noticeable drop at the halfway point.

These results are very interesting in a number of ways. First, we see that some of the humans, after a small number of training episodes, have created an impressive agent, with an average episode reward well over any systems we have seen. Most of the averages are near

65.0, which indicates the agent placed 4 Frogs in the home and either placed a fifth Frog in the home or died along the way. This performance is exceptionally well within Frogger. However, as training continues, we see a noticeable drop in almost all trials in the average episode reward. This halfway point marks when the epsilon hyperparameter is halved, in order to promote exploitation of the policy and no more exploration. The surprising point about this fact is that the average episode reward begins to near the average of all other systems we have seen thus far, placing roughly 2 Frogs in the home on an average run. In order to continue to investigate these findings more, we decided to run the human results for another 5000 episodes, thus extending the point when epsilon is halved and to investigate if more training will bring the average episode reward back up to where it was at the end of human training. Figure 7.2 provides these results.

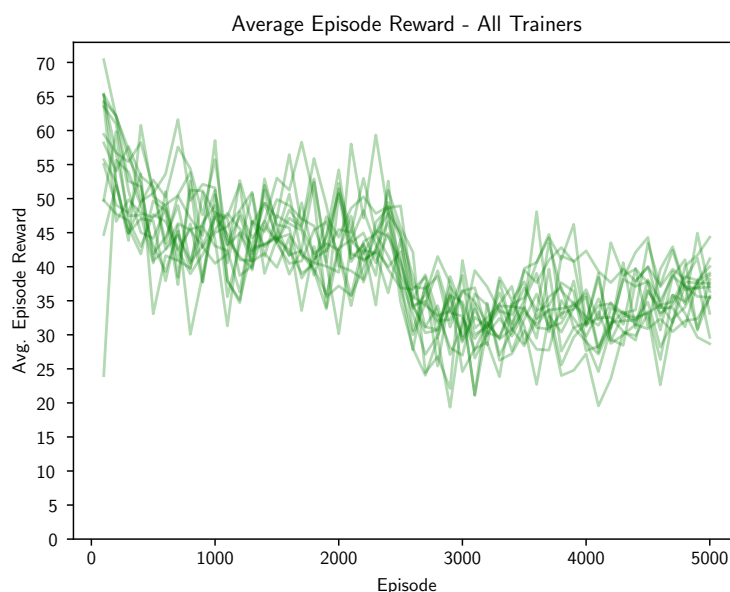


Figure 7.2: The average episode reward averaged over all 15 human trials after training for an additional 5000 episodes. Again, a noticeable drop in the average episode reward is evident in the halfway point.

Again, these results show similar results to the continuation training of 2000 episodes. We see the drop in average episode reward again at the halfway point, or when epsilon is

halved. There are a number of possible explanations to this phenomenon of dropping average reward:

- Since the human presented rare, and even some impossible situations, the exploration was finished from the agent’s perspective and the addition of random actions to the human’s developed weights actually began to reinforce negative behavior.
- The human developed a generalized solution with respect to the feature set and utilizing a standard Q-Learning algorithm began to revert this generalized solution back to the original Q-Learning agent’s ability at the start of training, thus reinforcing the original weights.
- The agent began to learn a cyclic path through the space, thus taking the local information and determining global information from that representation.
- The human trainers did not know enough about the underlying representation and thus failed to properly assist the agent in exploration.

We will touch on each of these possible explanations for the results. First, we will discuss the teachers and their individual results. To compare the results, we took each of the 15 teachers and wanted to compare the average episode reward immediately after training and at the 1000 episode mark in the continuation training (from the 2000 episode run). Table 7.1 provides this comparison.

Interestingly, the human trainers that had the best results at the end of human training had the largest decrease in episode reward at the 1000 episode mark. Some even had worse average episode reward than the Q-Learning agent baseline. Again, a number of explanations could be provided for these results. On a basic level, it is possible the teachers that provided unique and differing states had the best results following human training. This naturally leads to an exploration difference between these teachers and those that created vanilla and “boring” states. Since these teachers inherently presented unique states, at that point in time, the agent was finished with exploration. Thus, the addition of epsilon in standard training essentially “broke” the human’s work developing the weights. There is no substantial

Trainer	After Human Training	After 1000 Episodes	Increase or Decrease? (I/D)
A	0.0	16.7	I
B	0.0	36.4	I
C	20.6	68.5	I
D	22.1	15.4	D
E	26.1	70.4	I
F	29.5	40.3	I
G	37.5	55.0	I
H	43.6	16.7	D
I	52.9	16.7	D
J	54.7	63.7	I
K	55.3	17.5	D
L	55.8	55.3	D
M	59.7	17.4	D

Table 7.1: The individual results for trainers in the 2000 episode continuation training. This table presents the average episode reward for each trainer immediately after human training and after 1000 episodes (or the point when epsilon decayed first) while continuing training with a standard Q-Learning agent.

evidence of this difference and it seems this explanation does not provide sufficient reason why Curriculum Development did not work as expected. Some trainers even broke the developed weights at the start of human training, by garnering 0.0 average reward, or less than the standard Q-Learning average. However, two of those trainers (C and E) broke their original agents, but ended up having the best average episode rewards we’ve seen in this report. 70.4 for trainer E is incredibly high, essentially always placing 3 Frogs in the home, with some runs placing 4 or even all 5 Frogs in their homes.

The difference in trainers also points to another possible reasoning for the negative results. The human trainers may not have had enough knowledge, information, or understanding of the underlying representation of the agent. By not knowing how RL works and the local feature representation, the human’s could not provide the best scenarios that would help reinforce and build a strong feature set. It would be interesting to test with human experts that have RL knowledge and can understand the underlying representation to provide the agent with the best scenarios. This unfortunately does not point to the explanation of the significant drop in average episode reward throughout the continued training.

Another possible explanation is that the human trainers helped the agent learn a generalized solution to Frogger and the introduction of the standard Q-Learning agent eventually destroyed this generalized solution and reverted (or reinforced) the original solution that focused on memorizing paths. When viewing a developed agent in testing mode (only exploiting policy), the agent seems to follow almost the same path to the homes each time. This cyclic path continues to be reinforced throughout training and it's apparent the agent has learned the exact path while exploiting the policy. This means the agent used its local representation and learned something about the global state of the world. Due to the cycling of the cars, logs, and turtles and the constant speed of these objects, the agent can tell which path to take and which home to aim for based on the position of the cars. In the end, the agent did not truly learn a generalized solution to the problem, but learned global information about the state space from its limited, local representation. If true this would be a surprising but unintended result. This insight possibly explains the results at the end of human training. At that point, the cyclic path and global view was inherently broken by the human's exploration states created. Thus, the agent took well developed and converged weights and shocked them. That shock ended up taking the developed weights and presenting them into unique, new states. These new states forced the agent to change its knowledge of the global state, as it did not always start at the same place and it did not have the same repeated states presented to it. Arguably, this means we should take the best developed agents at the end of human training and utilize those as our Frogger agent, as that means the agent will have a generalized solution to the problem and be able to react to any situation presented to it.

Overall, Curriculum Development exposed a tremendous amount of information about the underlying agent operating in the state space. The results we received from Curriculum Development allowed us to see that the agent, in every system, was behaving in an unintended way. The system did not work exactly as intended (to generate a better policy following training) in *some* instances and it did in others. All instances, however, ended up reverting back to rewards similar to the other systems, suggesting that the cyclic paths discussed became reinforced with additional training and the agent returned to its original behavior.

Regardless of the unexpected results seen in Curriculum Development, we learned more about the underlying representation and how that can be utilized to design a better system. Curriculum Development worked in ways it was not intended to, but still worked on some level. It still had some agents producing impressive results after human training, proving that new exploration and unique state space manipulation can lead to a well-developed policy. A number of possible expansions and modifications of the system could lead to better performing agents after training. With further investigation, we could find the proper utilization of the system to see if we can utilize the human training to truly develop a generalized agent for Frogger.

Chapter 8

Future Work

Upon review of the project, we still have some questions that could bring insight to the systems as a whole, as well as some other experiments we would like to test to see if those other experiments will help increase the positive results of the systems as they currently stand. Especially for Curriculum Development, we see a large number of possible experiments to try in the future.

The possible expansions we seek to implement are:

1. Provide human trainers the ability to learn more about Frogger by playing the standard game first
2. Invite RL experts who understand the underlying representation to interact with the systems
3. Bootstrap the computer agents to determine if the agent improves each time with a better teacher
4. Combine both Time Warp and Curriculum Development into one system
5. Use a less-developed set of weights at the start of Curriculum Development
6. Further examine the states created by the human trainers in Curriculum Development to determine which types of states led to better results

7. Modify the state representation of Frogger to include more global features in the representation for both the Time Warp and Curriculum Development systems
8. Further examine differences in teaching strategy for Time Warp to determine which strategies led to better results, this also includes examining the differences between the good and bad teachers after continuing training with a standard Q-Learning agent
9. Varying the start state for Time Warp to avoid the learning of cyclic paths through the environment

First, we'd like to provide human trainers with better training in the Frogger domain. We noticed many of the human trainers did not have a full understanding of the game at the start of training. We propose having the human trainers play a few rounds of the standard Frogger game first, to ensure they understand the problem domain as a whole. This will avoid them wasting valuable training time to just learn how to play the game. Although the difference between "good" and "bad" trainers did not truly effect the overall reward average, we'd like to investigate how better Frogger players perform with the system. Next, we'd like to invite RL experts, or human trainers that understand RL on a deep level, to interact with both systems. RL experts may provide better scenarios (in Curriculum Development) and better strategies (in Time Warp) because of their inherent understanding of RL and that could produce better performing agents in the end.

Another experiment we would like to attempt involves the computer agent teachers developed by Torrey and Taylor and Amir *et al.* As seen in the results of the computer agent teachers, each of the systems eventually developed a better policy than the teacher used to assist the new agent. We hypothesize that by "bootstrapping" the teachers, we could continually develop better agents. This would mean that once we develop a policy for an agent, we use that agent as the teacher to assist the training of another agent. While this may take a longer time to develop a policy in Frogger, we believe it would eventually create a much better policy.

Next, we'd like to combine both the Time Warp and Curriculum Development systems. Effectively, the human trainer would set up a scenario like in Curriculum Development, then

immediately be handed control of the agent like in Time Warp. We hypothesize this will allow for an even better policy to be developed as the human will have **complete** control of the agent’s exploration throughout the space. This method will be incredibly similar to the RL sub-field Learning from Demonstration (LfD). Since the human will essentially set up situations and then have control, the human will essentially be demonstrating the correct policy. However, the main difference between this combination and LfD will be that the human can give control back to the agent to learn on its own and only provide actions when the agent is in important states.

Then, we’d like to also use a less-developed set of wights that each human trainer started with in Curriculum Development. It is possible the weights provided were too well reinforced for the type of exploration the humans started. With less-developed weights, we can place more control to the human by allowing the human to manipulate the weights more. If the weights are too developed, the human’s scenarios may not have enough reinforcement to substantially change the weights. This could lead to the development of a solution that is more generalized for Frogger.

Next, we’d like to further examine the states created by the human trainers in Curriculum Development. This could provide more insight into the results of each individual and what types of states led to better results in Curriculum Development.

For both systems, we’d also like to modify the state space we used for Frogger. We desire to create a more global representation of Frogger rather than a purely local representation. This difference could mitigate the learning of cyclic paths in Frogger, which would also modify the common convergence point of all agents. A global representation may also allow the agents to learn a generalized solution to Frogger, which would result in better performing agents.

Similar to our examination of the states created in Curriculum Development, we would also like to examine the teaching strategies for the Time Warp subjects. By extracting the strategies from each subject, we could find the strategies that led to the best performing agents. Additionally, we would like to investigate the strategies between the truly good and bad trainers (after continuing training with a standard Q-Learning agent). We could not

find a correlating variable between these two groups and would like to find what strategies led to separating the trainers into these distinct groups.

Lastly, we'd like to vary the start state that the agent begins in with the Time Warp system. Specifically, we'd like to vary the x-location of the Frog at the beginning of an episode. We hypothesize this variation of start state will also mitigate the learning of cyclic paths through the state space and lead to a generalized solution following human training.

Chapter 9

Conclusions

In this project, we examined two novel systems that incorporate a human expert to train a RL agent. These new systems were able to properly integrate a human trainer into a standard RL system and produce meaningful results. Time Warp and Curriculum Development provide an interface that assists the agent in proper exploration of the state space. We showed that by manipulating the exploration of an agent with a human teacher works just as well as the computer agent teaching systems. Additionally, the human expert has assisted the agent in satisfying the planning requirement in a complex environment such as Frogger. Our first hypothesis of this project was that integrating a human teacher to assist in the exploration of the space would help the agent create a better policy at a faster rate than Q-Learning alone. This hypothesis was upheld through Time Warp. We proved that a human teacher can be added to an RL training system such that the development of an optimal policy for an agent can be faster and more correct than with a standard agent alone. Our other hypothesis, which sought to solve the planning requirement apparent in complex environments such as Frogger, was that by allowing a human expert to present a number of scenarios for the agent to train on the agent could produce a better optimal policy. While Curriculum Development sought to solve this planning requirement, we did not fully prove our hypothesis. Instead, we learned a great deal about the underlying representation of the agent, while still producing some results that did help support our hypothesis.

This paper first introduced the problem we solved: the ability to include a human teacher

into the training of an RL agent. Then, we explained the necessary background of Q-Learning and forms of assistance to RL. Following that discussion, we implemented the most successful computer teaching agents in the Frogger domain. This implementation not only provided a baseline to compare our systems with, but also further proved the original authors' work by extending the same principles to a more complex domain. Next, we introduced Time Warp and demonstrated Time Warp's ability to not only integrate a human teacher into the system, but also develop a better optimal policy at a faster rate than a Q-Learning agent alone. This system also matched the computer teaching systems' results. Then, we introduced our last system, Curriculum Development. This system produced the most interesting results. Curriculum Development exposed some inherent characteristics about the underlying agent. We found some trainers were able to help the agent learn a generalized solution to the problem, which produced a well-performing agent. However, after some training with a standard Q-Learning algorithm, the agent reverted back to the old weights used at the beginning of human training and converged to similar results as the other systems tested in this project. Curriculum Development developed some outstanding agents at very specific and random points in the training following human interaction. This means it is possible to utilize Curriculum Development to produce better policies, but it did not prove a standardized and specific procedure needed to perform this training on a large scale. With further investigation, some of the possible explanations for these results could be found and corrected to produce great results on a large scale. Finally, we discussed the future work that could be used to further expand on our findings in this project. Our future work section has a lot of potential to continue to develop human assistance systems and provide a meaningful and straightforward way to integrate humans into RL training.

Overall, this project produced great results that provided a tremendous amount of insight into our system design and proved that humans introduced into a Q-Learning system to assist in exploration can both create a better policy and find that policy at a faster rate than standard RL agents alone.

Bibliography

- [1] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara Grosz. Interactive teaching strategies for agent training. In *International Joint Conference on Artificial Intelligence*, New York City, USA, 2016.
- [2] Matthew Emigh, Evan Kriminger, Austin Brockmeier, José Príncipe, and Panos Pardalos. Reinforcement learning in video games using nearest neighbor interpolation and metric learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 8:56–66, 2016.
- [3] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *The Fifth International Conference on Knowledge Capture*, September 2009.
- [4] W. Bradley Knox and Peter Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, May 2010.
- [5] W. Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.
- [6] Poole David L. and Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, New York, NY, USA, 2010.
- [7] Richard Maclin and Jude W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1):251–281, Mar 1996.
- [8] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.
- [9] David L. Moreno, Carlos V. Regueiro, Roberto Iglesias, and Senn Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. In *Proceedings of Towards Autonomous Robotics Systems 2004. Univ. of Essex*, 2004.
- [10] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, Singapore, May 2016.

- [11] padorange. pyfrogger. <https://sourceforge.net/projects/pyfrogger/>, 2013.
- [12] Peter Stone Sanmit Narvekar, Jivko Sinapov. Autonomous task sequencing for customized curriculum design in reinforcement learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2536–2542, 2017.
- [13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.
- [14] Norman Tasfi. Pygame learning environment. <https://github.com/ntasfi/PyGame-Learning-Environment>, 2016.
- [15] Andrea L. Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, AAAI’06*, pages 1000–1005. AAAI Press, 2006.
- [16] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS ’13*, pages 1053–1060, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

Appendix A

Script Provided to Subjects

In this appendix, we will provide the script given to each group of subjects before interacting with both the Time Warp and Curriculum Development systems. These scripts did have deviations if subjects asked questions during the briefing period, but mostly capture what was communicated to the subjects before they began training.

A.1 Time Warp Script

“Reinforcement learning is a technique used to teach a computer program (agent) the right way to act in a given environment. This is done by a trial-and-error process. It is very similar to training a dog. If the dog waits by the door to go outside, then we give the dog a treat giving positive reward. If the dog instead goes over to the carpet and ruins the carpet, we scold the dog giving negative reward. The first step to reinforcement learning is that we build a mathematical model of the domain, which in our case is Frogger. We define what is good and what is bad in the domain. Then, we use a reinforcement learning program that will explore the domain and build a mathematical function to determine the best actions to take. The idea is that the program will build a function that gives the agent as much positive reward as possible.

“As great as reinforcement learning is, it does have a few downfalls. It takes a long time to get a well-working reinforcement learning program. Also, sometimes the program cannot

find the best moves, but just pretty good moves. Both of these problems can be solved, however with a new approach.

“Just like in academics, a teacher can assist a reinforcement learning ‘student’ in finding the best moves quicker. Current approaches use a reinforcement learning program that already knows the best moves to guide the student along, but this defeats the purpose. There is no reason to have another program be taught the best moves if one already exists. We desire to have a human teacher that knows the best moves to interact with the system and convey this knowledge to the agent.

“The system you will interact with is called Time Warp. You will observe an agent training and when you believe the agent did something wrong, you can reverse time and provide the correct sequence of actions. Here’s how you use Time Warp.

“When you see the agent do something wrong, press the space bar to freeze training. Then, use the up and down arrow keys to cycle through the possible frames you can revert back to. Once you find the frame you want to resume training, press the space bar again. Then, you have full control of the agent! You will use the up/down/left/right arrow keys to provide actions to the agent. If you’d like the agent to resume training on its own, press the space bar again. You will have a time limit of ten minutes and a budget of 5,000 possible actions. Please note, when you do not provide an action on the keyboard, that will automatically provide the ‘do nothing’ action to the agent and still count against your budget. You also are able to quit at any time by pressing Q on the keyboard.

“There are a number of strategies you can employ while training, but we will give you the two extremes and you can select your own strategy that lies at the extremes or in the middle. One extreme is that you immediately take control of the agent and simply play Frogger. Your budget will run out quicker, but you’ll have direct control of the agent. The other extreme is that you only provide one action every time you freeze training. This approach will take much longer to train, but is a possible strategy to training. Just as a final reminder, you can also use a mixture of these two extremes. You can watch the agent train for awhile, stop and provide some actions, then hand training back over. It’s completely up to you!”

At this point, we provided the subjects with a demo of the system to reinforce the controls

and what the domain looked like.

A.2 Curriculum Development Script

“Reinforcement learning is a technique used to teach a computer program (agent) the right way to act in a given environment. This is done by a trial-and-error process. It is very similar to training a dog. If the dog waits by the door to go outside, then we give the dog a treat giving positive reward. If the dog instead goes over to the carpet and ruins the carpet, we scold the dog giving negative reward. The first step to reinforcement learning is that we build a mathematical model of the domain, which in our case is Frogger. We define what is good and what is bad in the domain. Then, we use a reinforcement learning program that will explore the domain and build a mathematical function to determine the best actions to take. The idea is that the program will build a function that gives the agent as much positive reward as possible.

“As great as reinforcement learning is, it does have a few downfalls. It takes a long time to get a well-working reinforcement learning program. Also, sometimes the program cannot find the best moves, but just pretty good moves. Both of these problems can be solved, however with a new approach.

“Just like in academics, a teacher can assist a reinforcement learning ‘student’ in finding the best moves quicker. Teachers typically will create a curriculum, or a schedule of topics to cover over a given period, to help teach their students. By providing a number of topics for a reinforcement learning agent to learn, we can possibly help the agent learn a better solution to the task at hand.

“The system you will interact with is called Curriculum Development. You will create a number of scenarios to present to the agent and the agent will train on those scenarios. Your goal is to make a better agent than what is provided to you at the start of using the system. You will do this by presenting scenarios. The idea is to make scenarios that you think are important for the Frog to learn how to accomplish. Pretend you’re instructing a child on what they need to know to do well at Frogger and present those same situations to

the Frog. Now, we'll teach you how to use Curriculum Development.

“You'll be presented with a starting game view of Frogger. As you develop scenarios, the last scenario you developed will be loaded each round. You can click the rows to select the row you'd like to manipulate. You can manipulate all rows of cars, logs, and turtles. Place them in the position you'd like them. To move the objects, use the left/right arrows. They will move along their row as if they were actually running in the game. When you want to switch between selecting the rows of objects and placing the Frog's position, press R. The game will say you are in 'setting Frog' mode. Click on the screen to place the Frog. The Frog will only go to positions that are legal when training begins. Again, you can alternate between setting the rows and the Frog by pressing the R button on the keyboard. When you have set up your scenario, press the space bar to begin training. The agent will run through one episode of training. One episode is either placing Frogs in all five homes or dying. When the episode is over, your last scenario will be loaded and you can develop another scenario.

“You will have ten minutes to set up situations. You can also quit at any time by pressing Q on the keyboard. Be creative with your scenarios or don't be creative! Just create situations that you believe are important for the Frog to learn.”

At this point, we provided the subjects with a demo of the system to reinforce the controls and what the domain looked like when they began training.